305-CD-043-001

EOSDIS Core System Project

Flight Operations Segment (FOS) Resource Management Design Specification for the ECS Project

October 1995

Hughes Information Technology Corporation
Upper Marlboro, MD

Flight Operations Segment (FOS) Resource Management Design Specification for the ECS Project

October 1995

Prepared Under Contract NAS5-60000
CDRL Item #046

APPROVED BY

Cal Moore /s/ 9/22/95

Cal Moore, FOS CCB Chairman Date
EOSDIS Core System Project

Hughes Information Technology Corporation
Upper Marlboro, Maryland

This page intentionally left blank.

Preface

This document, one of nineteen, comprises the detailed design specification of the FOS subsystems for Releases A and B of the ECS project. This includes the FOS design to support the AM-1 launch.

The FOS subsystem design specification documents for Releases A and B of the ECS project include:

305-CD-040	FOS Design Specification (Segment Level Design)
305-CD-041	Planning and Scheduling Design Specification
305-CD-042	Command Management Design Specification
305-CD-043	Resource Management Design Specification
305-CD-044	Telemetry Design Specification
305-CD-045	Command Design Specification
305-CD-046	Real-Time Contact Management Design Specification
305-CD-047	Analysis Design Specification
305-CD-048	User Interface Design Specification
305-CD-049	Data Management Design Specification
305-CD-050	Planning and Scheduling Program Design Language (PDL)
305-CD-051	Command Management PDL
305-CD-052	Resource Management PDL
305-CD-053	Telemetry PDL
305-CD-054	Real-Time Contact Management PDL
305-CD-055	Analysis PDL
305-CD-056	User Interface PDL
305-CD-057	Data Management PDL
305-CD-058	Command PDL

Object models presented in this document have been exported directly from CASE tools and in some cases contain too much detail to be easily readable within hard copy page constraints. The reader is encouraged to view these drawings on line using the Portable Document Format (PDF) electronic copy available via the ECS Data Handling System (EDHS) at URL http://edhs1.gsfc.nasa.gov.

This document is a contract deliverable with an approval code 2. As such, it does not require formal Government approval, however, the Government reserves the right to request changes within 45 days of the initial submittal. Once approved, contractor changes to this document are handled in accordance with Class I and Class II change control requirements described in the EOS Configuration Management Plan, and changes to this document shall be made by document change notice (DCN) or by complete revision.

Any questions should be addressed to:

Data Management Office
The ECS Project Office
Hughes Information Technology Corporation
1616 McCormick Drive
Upper Marlboro, MD 20774

Abstract

The FOS Design Specification consists of a set of 19 documents that define the FOS detailed design. The first document, the FOS Segment Level Design, provides an overview of the FOS segment design, the architecture, and analyses and trades. The next nine documents provide the detailed design for each of the nine FOS subsystems. The last nine documents provide the PDL for the nine FOS subsystems. It also allocates the level 4 FOS requirements to the subsystem design.

Keywords: FOS, design, specification, analysis, IST, EOC

This page intentionally left blank.

Change Information Page

List of Effective Pages	
Page Number	**Issue**
Title	Original
iii through xii	Original
1 -1 and 1-2	Original
2-1 through 2-4	Original
3-1 through 3-148	Original
AB-1 through AB-4	Original
GL-1 through GL-10	Original

Document History			
Document Number	**Status/Issue**	**Publication Date**	**CCR Number**
305-CD-043-001	Original	October 1995	95-0652

This page intentionally left blank.

Contents

Abbreviations and Acronyms

Glossary

Figures

Tables

Abbreviations and Acronyms

Glossary

This page intentionally left blank.

1. Introduction

1.1 Identification

The contents of this document defines the design specification for the Flight Operations Segment (FOS). Thus, this document addresses the Data Item Description (DID) for CDRL item 046 305/DV2 under Contract NAS5-60000.

1.2 Scope

The Flight Operations Segment (FOS) Design Specification defines the detailed design of the FOS. It allocates the level 4 FOS requirements to the subsystem design. It also defines the FOS architectural design. In particular, this document addresses the Data Item Description (DID) for CDRL # 046, the Segment Design Specification.

This document reflects the August 23, 1995 Technical Baseline maintained by the contractor configuration control board in accordance with ECS Technical Direction No. 11, dated December 6, 1994. It covers releases A and B for FOS. This corresponds to the design to support the AM-1 launch.

1.3 Purpose

The FOS Design Specification consists of a set of 19 documents that define the FOS detailed design. The first document, the FOS Segment Level Design, provides an overview of the FOS segment design, the architecture, and analyses and trades. The next nine documents provide the detailed design for each of the nine FOS subsystems. The last nine documents provide the PDL for the nine FOS subsystems.

1.4 Status and Schedule

This submittal of DID 305/DV2 incorporates the FOS detailed design performed during the Critical Design Review (CDR) time frame. This document is under the ECS Project configuration control.

1.5 Document Organization

305-CD-040 contains the overview, the FOS segment models, the FOS architecture, and FOS analyses and trades performed during the design phase.

305-CD-041 contains the detailed design for Planning and Scheduling Design Specification.

305-CD-042 contains the detailed design for Command Management Design Specification.

305-CD-043 contains the detailed design for Resource Management Design Specification.

305-CD-044 contains the detailed design for Telemetry Design Specification.

305-CD-045 contains the detailed design for Command Design Specification.

305-CD-046 contains the detailed design for Real-Time Contact Management Design Specification.

305-CD-047 contains the detailed design for Analysis Design Specification.

305-CD-048 contains the detailed design for User Interface Design Specification.

305-CD-049 contains the detailed design for Data Management Design Specification.

305-CD-050 contains Planning and Scheduling PDL.

305-CD-051 contains Command Management PDL.

305-CD-052 contains Resource Management PDL.

305-CD-053 contains the Telemetry PDL.

305-CD-054 contains the Real-Time Contact Management PDL.

305-CD-055 contains the Analysis PDL.

305-CD-056 contains the User Interface PDL.

305-CD-057 contains the Data Management PDL.

305-CD-058 contains the Command PDL.

Appendix A of the first document contains the traceability between Level 4 Requirements and the design. The traceability maps the Level 4 requirements to the objects included in the subsystem object models.

Glossary contains the key terms that are included within this design specification.

Abbreviations and acronyms contains an alphabetized list of the definitions for abbreviations and acronyms used within this design specification.

2. Related Documentation

2.1 Parent Document

The parent documents are the documents from which this FOS Design Specification's scope and content are derived.

194-207-SE1-001	System Design Specification for the ECS Project
304-CD-001-002	Flight Operations Segment (FOS) Requirements Specification for the ECS Project, Volume 1: General Requirements
304-CD-004-002	Flight Operations Segment (FOS) Requirements Specification for the ECS Project, Volume 2: AM-1 Mission Specific

2.2 Applicable Documents

The following documents are referenced within this FOS Design Specification or are directly applicable, or contain policies or other directive matters that are binding upon the content of this volume.

194-219-SE1-020	Interface Requirements Document Between EOSDIS Core System (ECS) and NASA Institutional Support Systems
209-CD-002-002	Interface Control Document Between EOSDIS Core System (ECS) and ASTER Ground Data System, Preliminary
209-CD-003-002	Interface Control Document Between EOSDIS Core System (ECS) and the EOS-AM Project for AM-1 Spacecraft Analysis Software, Preliminary
209-CD-004-002	Data Format Control Document for the Earth Observing System (EOS) AM-1 Project Data Base, Preliminary
209-CD-025-001	ICD Between ECS and AM1 Project Spacecraft Software Development and Validation Facilities (SDVF)
311-CD-001-003	Flight Operations Segment (FOS) Database Design and Database Schema for the ECS Project
502-ICD-JPL/GSFC	Goddard Space Flight Center/MO&DSD, Interface Control Document Between the Jet Propulsion Laboratory and the Goddard Space Flight Center for GSFC Missions Using the Deep Space Network

530-ICD-NCCDS/MOC	Goddard Space Flight Center/MO&DSD, Interface Control Document Between the Goddard Space Flight Center Mission Operations Centers and the Network Control Center Data System
530-ICD-NCCDS/POCC	Goddard Space Flight Center/MO&DSD, Interface Control Document Between the Goddard Space Flight Center Payload Operations Control Centers and the Network Control Center Data System
530-DFCD-NCCDS/POCC	Goddard Space Flight Center/MO&DSD, Data Format control Document Between the Goddard Space Flight Center Payload Operations Control Centers and the Network Control Center Data System
540-041	Interface Control Document (ICD) Between the Earth Observing System (EOS) Communications (Ecom) and the EOS Operations Center (EOC), Review
560-EDOS-0230.0001	Goddard Space Flight Center/MO&DSD, Earth Observing System (EOS) Data and Operations System (EDOS) Data Format Requirements Document (DFRD)
ICD-106	Martin Marietta Corporation, Interface Control Document (ICD) Data Format Control Book for EOS-AM Spacecraft
none	Goddard Space Flight Center, Earth Observing System (EOS) AM-1 Flight Dynamics Facility (FDF) / EOS Operations Center (EOC) Interface Control Document

2.3 Information Documents

2.3.1 Information Document Referenced

The following documents are referenced herein and, amplify or clarify the information presented in this document. These documents are not binding on the content of this FOS Design Specification.

194-201-SE1-001	Systems Engineering Plan for the ECS Project
194-202-SE1-001	Standards and Procedures for the ECS Project
193-208-SE1-001	Methodology for Definition of External Interfaces for the ECS Project
308-CD-001-004	Software Development Plan for the ECS Project
194-501-PA1-001	Performance Assurance Implementation Plan for the ECS Project
194-502-PA1-001	Contractor's Practices & Procedures Referenced in the PAIP for the ECS Project
604-CD-001-004	Operations Concept for the ECS Project: Part 1-- ECS Overview, 6/95
604-CD-002-001	Operations Concept for the ECS project: Part 2B -- ECS Release B, Annotated Outline, 3/95

604-CD-003-001	ECS Operations Concept for the ECS Project: Part 2A -- ECS Release A, Final, 7/95
194-WP-912-001	EOC/ICC Trade Study Report for the ECS Project, Working Paper
194-WP-913-003	User Environment Definition for the ECS Project, Working Paper
194-WP-920-001	An Evaluation of OASIS-CC for Use in the FOS, Working Paper
194-TP-285-001	ECS Glossary of Terms
222-TP-003-006	Release Plan Content Description
none	Hughes Information Technology Company, Technical Proposal for the EOSDIS Core System (ECS), Best and Final Offer
560-EDOS-0211.0001	Goddard Space Flight Center, Interface Requirements Document (IRD) Between the Earth Observing System (EOS) Data and Operations System (EDOS), and the EOS Ground System (EGS) Elements, Preliminary
NHB 2410.9A	NASA Hand Book: Security, Logistics and Industry Relations Division, NASA Security Office: Automated Information Security Handbook

This page intentionally left blank.

3. Real-Time Resource Management Subsystem

The Real-Time Resource Management Subsystem (RMS) is one of four real-time subsystems that reside on the Real-Time Servers (RTS) and/or the User Workstations within the EOC. Select real-time subsystems will also be included in the Instrument Support Toolkit (IST) that is executed at remote locations outside of the EOC. The RMS resides on both of the EOC hosts as well as within the IST and provides management and control functions for the remaining real-time subsystems. The real-time Server provides a single location where real-time telemetry monitoring occurs. The command processing is coupled with this monitoring due to the dependency of telemetry processing for command and telemetry verification. The real-time architecture, facilitated by the RMS, is based on providing logical strings, which manage the real-time resources. A logical string is a collection of hardware and software resources, and information about how these resources are being used within the EOC, to provide spacecraft and instrument control and monitoring during real-time contacts, simulations, and historical replays. A unique logical string exists for each real-time scenario (i.e., contact, simulation, and historical replay). There are five attributes of a logical string that make it unique. These attributes include:

- a spacecraft identifier which marks a logical string for support of a specific mission
- a operational database identifier which indicates the database version used in configuration of the software in a specific logical string
- the data source which indicates the origin of the telemetry data being monitored within the logical string (i.e., real-time, simulation, historical replay)
- the mode attribute which indicates the intended use of the logical string (i.e., operational, test, training)
- and the state attribute which indicates if a logical string in the operational mode is actively performing its intended function (active), or if it exists only to perform its intended function in the event of a hardware or software failure in the active string (backup)

The number and type of resources managed within a given logical string can vary based on the characteristics listed above. For example, a historical replay logical string requires only real-time Telemetry processes for the purpose of decommutating the historical data, while real-time, operational logical strings that monitor real-time contacts require a full complement of Telemetry, Command and Real-time Contact Management subsystem processes as well as FUI Ground Script Controller, DMS Archiver and Parameter Server processes.

Of the four real-time subsystems, the Resource Management Subsystem (RMS) is the only one that is considered permanent or persistent. In other words, as long as there is a real-time server or user workstation in operation within the EOC, or an IST running outside of the EOC, the RMS is available to provide service to its users. The RMS provides a level of control and management for the real-time subsystems as well as additional select subsystem processes that provide a mission critical function within the EOC. The RMS is responsible for providing real-time software resources and failure recovery capabilities in response to user requests.

One of the more important functions of the Resource Management Subsystem is that of privilege

management. Command Authority and Ground Control Privilege are designations bestowed on users to act in critical roles within the EOC. Command Authority is granted to one EOC user per command destination for the purpose of sending real-time commands to a spacecraft. This privilege is managed within a logical string to ensure that there is a single point of command for an EOC spacecraft. The Ground Control Privilege is granted to one EOC user per logical string for the purpose of modifying the ground configuration of the hardware and software resources within that logical string. These privileges are granted only to users that are pre-authorized by the Flight Operations Team to perform in these roles. Further, these privileges are granted only to users signed on to User Workstations within the EOC.

The design documentation that follows describes two types of RMS subsystems, the Real-Time Server RMS and the Workstation RMS . These subsystems are named according to where they reside and differ in that they serve complementary functions. The Real-Time Server RMS is made up of two executable processes, the String Manager and the Resource Monitor. The Workstation RMS employs only the String Manager process. These subsystems work together to provide the EOC user with control of real-time resources as well as global visibility into the real-time activities within the EOC.

Users of the EOC wishing to create a logical string to monitor a real-time contact will allocate resources on a Real-Time Server in order to allow multiple users to simultaneously connect to, or access that information from different workstations. The string that is created is termed a shared service or shared logical string. In this scenario, a logical string is created on a Real-Time Server by a RTS RMS String Manager process. This action is taken in response to a request received by a Workstation RMS String Manager process and forwarded to the Real-Time Server. Information is returned to the requesting workstation after the new logical string is created, and the Workstation RMS, in turn, provides that information to the user display. This scenario is discussed in further detail in sections 3.2.4.3 and 3.2.4.4 of this document.

Users of the EOC can also request dedicated services or dedicated logical strings. An example of a dedicated logical string is that of a dedicated historical replay. In this scenario, a user would simply request the dedicated service, and resources that reside only on that user's workstation would be employed to provide the requested service. Other users would not have visibility into, or knowledge of that user's activities, nor would other users be able to connect to, or otherwise access that activity. In other words, the activity is dedicated to a single EOC user. Shared replays, using shared logical strings can also be established for multiple users. The dedicated replay scenario is discussed in further detail in section 3.2.4.11 of this document.

When a logical string exists on a Real-Time Server it is shareable by definition. When an EOC user connects to a logical string, that user is provided a set of telemetry processes to execute on his local workstation, equal in number to those that are executing within the selected logical string on the Real-Time Server. The user must decide at connect time if he wants to control the ground configuration of the processes that are running on his local workstation or simply take the ground configuration and of the processes running on the Real-Time Server. To make this selection, the user must connect to the logical string in one of two ways: by mirrored connection or by tailored connection. These connection methods are described in the paragraphs that follow.

When a user requests a mirrored connection to a logical string, that user is provided the same telemetry ground configuration as the telemetry processes running on the Real-Time Server.

Changes to the ground configuration on the Real-Time Server can only be made by a single user per logical string with the Ground Control Privilege. After a mirrored user connects to a logical string, all subsequent changes to the ground configuration of the logical string are made to the telemetry processes on the user's local workstation as well. Thus, the ground configurations of the telemetry processes on the Real-Time Server and those that execute on the User Workstation are synchronized. A benefit of the mirrored connection is that the user of this connection is eligible to request Ground Control Privilege for the logical string to which he is connected. If this user is pre-authorized for that privilege, he can potentially control the ground configuration of the telemetry processes that execute on the Real-Time Server for this logical string. It is not required, however, that the mirrored user serve in this capacity. Mirrored logical string connections will be the nominal connection type of choice among EOC users.

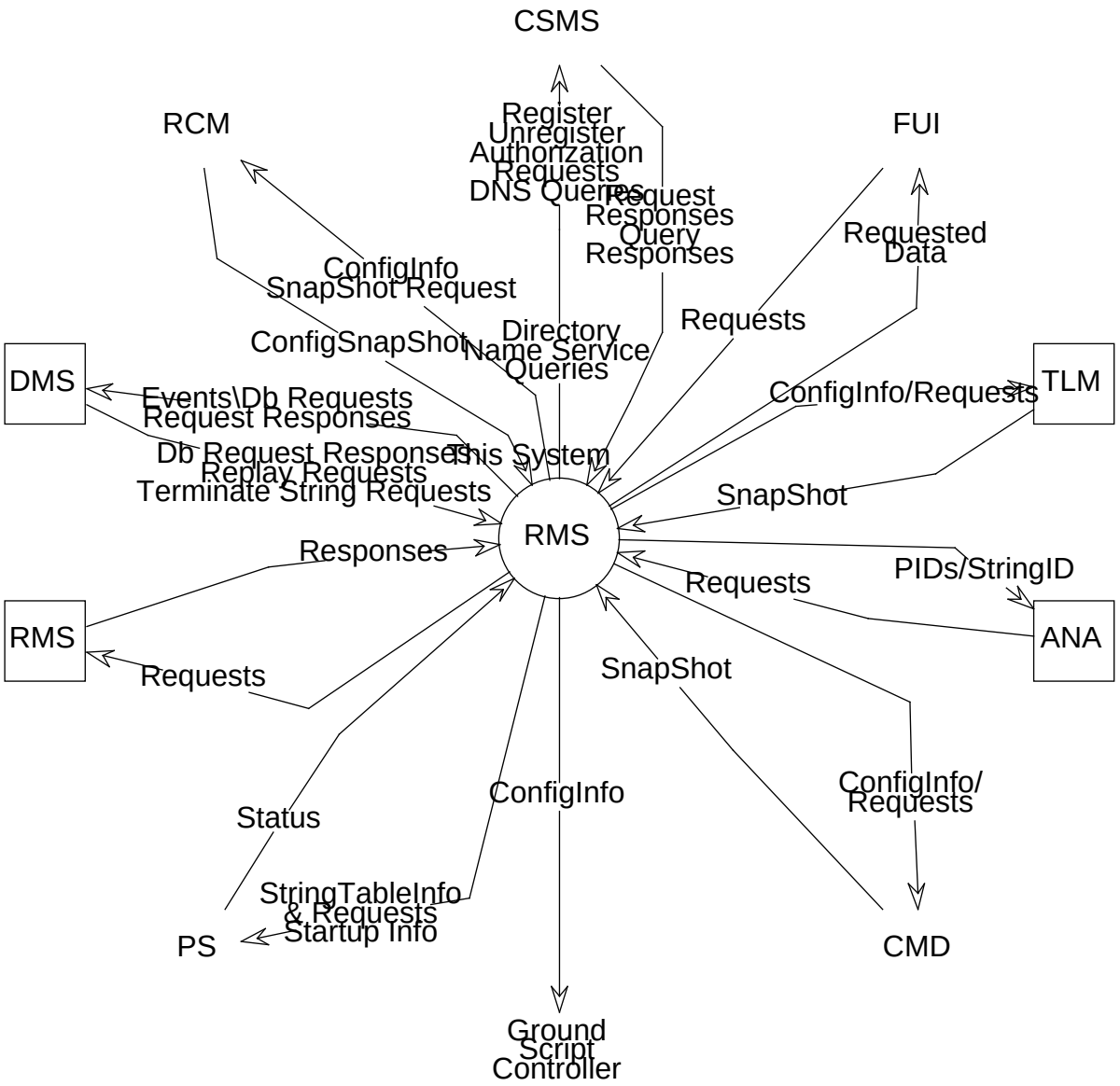
When a user requests a tailored connection to a logical string, that user is initially provided the same telemetry ground configuration as the telemetry processes running on the Real-Time Server. Upon connection however, the tailored user controls the ground configuration for the telemetry processes created on his local workstation independent of any privilege. Changes made by this user affect his local processes only. Further, this user is not eligible to gain the Ground Control Privilege that would allow him to modify the ground configuration of the telemetry processes running on the Real-Time Server for this particular logical string. Tailored logical string connection will be the nominal connection type of choice among IST users.

Ground Control Privilege, much like Command Authority, is requested by EOC users and granted by the RMS to users that are pre-authorized by the Flight Operations Team to have this privilege for the given mission supported in a given logical string.

3.1 Resource Management Subsystem Context

Upon system startup of a Real-Time Server, the Resource Management Subsystem will request and receive a default configuration procedure from the Data Management Subsystem. The default configuration file will convey to a given RMS, information about the real-time spacecraft contacts and backup processing for which it will be responsible. This file will convey to the RMS the same information that a user entering a real-time service request would provide. This allows the same RMS software to be used to respond to the service request whether the origin is an EOC user or default configuration file.

When a logical string is requested by the FOS User Interface Subsystem and subsequently created by the Resource Management Subsystem, the supporting software processes will be created as needed and configured for the specific purpose they are to fulfill. Software processes created by the RMS include Telemetry, Command, and Real-Time Contact Management subsystems, as well as Archive, Ground Script Control and Parameter Server processes. Additional configuration information can be conveyed to these subsystems after system startup on an as needed basis. User directives from FUI can be sent to RMS, where they will be forwarded to the appropriate support subsystem for action. Directive completion status's will be returned through RMS and returned to FUI for display. Any information received by the support subsystems that modify the ground configuration, is also sent by RMS to any backup logical strings that may exist. In the case of real-time Telemetry processes, the configuration changes are also forwarded to the Telemetry processes of mirrored users. This allows RMS to ensure configuration synchronization between active and backup strings and between Telemetry processes on User Workstation and Real-Time Servers. In



3.1-1. Resource Management Subsystem Context Diagram

the event of hardware or software failure the user would be notified of the failure, and issue a request for string failover via a FUI request window. The backup string established prior to the failure will assist RMS in restoring user capabilities with minimal interruption.

The CSMS/SCDO Management Subsystem will be integrated with the DMS Event Handler to provide the capability to monitor the hardware and software components of the EOC for changes in operational status. When new software resources are employed by RMS, MSS functions are invoked to register the new resources for monitoring. When changes in resource status's are detected by the MSS Monitoring service, a management event is generated by the MSS service and forwarded to the DMS Event Handler. The Parameter Server is informed of selected status changes via a proxy interface with the Event Handler. The Parameter Server in turn provides information about status parameters to the RMS and any other FOS application that registers an interest.

Both the Data Management and Analysis Subsystems have interface proxies from RMS for the purpose of sending requests for historical telemetry replay. Upon request, RMS provides telemetry subsystem resource(s) on a RTS or User Workstation to facilitate processing of historical telemetry. Requests for this service will originate from the user via the Data Management Subsystem if the user wishes to monitor the replay, or via the Analysis Subsystem if the user wishes to have off-line analysis performed on the historical data. In either case, the RMS provides an identical service of telemetry parameters to the Parameter Server allocated for the request. The Parameter Server, in turn serves the desired data upon request by the end-user subsystem.

3.2 RMS String Manager Component

The Resource Management Subsystem String Manager Component is designed to be installed and executed on the Real-Time Servers and all User Workstations within the EOC as well as all ISTs at remote locations outside of the EOC. The role that the RMS plays differs however, depending on its host.

When the String Manager Component is executed on a Real-Time Server, it serves several important functions. These functions include: responding to requests dealing with management of shared logical strings and user privileges; accessing the default configuration procedures and executing startup logical string requests; and communicating with the Resource Monitor Component (this is unique to the RTS RMS because the Resource Monitor Component is not executed on the User Workstations). The roles of the Real-Time Server RMS process are explained in the scenarios described in Section 3.2.4.1, .4, .6, .8, .10, .14, and .15.

The Workstation RMS String Manager processes serve a number of unique and important functions within the real-time architecture as well. These functions include: fielding and responding to user requests, from the FOS User Interface Subsystem, the Data Management Subsystem and the Off-line Analysis Subsystem; synchronizing telemetry configurations with backup logical strings and mirrored logical string connections; coordinating requests that affect multiple Real-Time Servers; granting and managing dedicated service requests; and employing CSMS/MSS services for user authorization lookups. The roles of the Workstation RMS process are explained in the String Table Parameters to Parameter Server scenarios described in Sections 3.2.4.1, .3, .5, .7, .9, .11, .12, and .13.

3.2.1 RMS String Manager Component Context

Upon execution of the RMS String Manager Component the first task is to determine what type of machine is host to the component. This is determined through a query to the CSMS/CSS Directory Naming Service. The host type determines the role that the String Manager Component is to play. These roles were described briefly in the preceding paragraphs in Section 3.2. The Workstation String Manager and Real-Time Server String Manager Component contexts are described in the paragraphs that follow and are illustrated in Figure 3.2.1-1.

The Workstation String Manager is responsible for receiving and responding to all user requests. Most user requests are received directly from the FOS User Interface Subsystem, while some dedicated service requests are received via the Data Management and Off-line Analysis subsystems. In cases where shared resources are involved, the Workstation String Manager is responsible for forwarding user requests to Real-Time Servers and coordinating activities between those servers (and their corresponding RTS String Manager Components) to ensure that configuration synchronization throughout the system is maintained.

When a user requests the Command or Ground Control privilege, the Workstation String Manager component has an additional responsibility. Before forwarding the privilege request to the RTS RMS for action, the Workstation String Manager first determines if the user is authorized to receive the privilege. A CSMS/CSS Authorization Service is queried with a privilege type and user identifier in order to determine if the requesting user is pre-authorized by the Flight Operations Team to serve in the requested capacity. If is user is authorized, the request will be forwarded to the RTS String Manager for action. Otherwise, a response is returned to the FUI process that generated the request and an event is generated indicating that the request was denied. (This request will be translated by the Data Management Subsystem into a management event that will be reported to the MSS Management Agent as a reportable security event.)

The Workstation String Manager is directly responsible for creating logical strings to support dedicated service requests and connection requests only. Therefore, the Telemetry subsystem is the only real-time subsystem that the Workstation RMS creates and configures in response to these service requests. Telemetry components are created in response to dedicated replay requests from the Data Management Queue Manager and the Off-line Analysis Request Manager, as well as in response to user connection requests to any shared logical string. Detailed scenarios describing the Workstation String Manager role in providing this service can be found in Sections 3.2.4.5 and 3.2.4.6.

If upon system startup the String Manager Component determines that it's host is a Real-Time Server, it's next task is that of accessing the database information and default configuration files that are stored by the Data Management Subsystem. The database information file provides startup configuration information for the Resource Management Subsystem. The default configuration file contains information about active missions that the EOC is supporting. The String Manager Component accesses this file and executes any startup logical string requests that may exist in that file for it's host system. This function allows for the monitoring of a real-time contact independent of an operator in the EOC with an active logical string connection.

Real-Time Server String Manager Components respond to user requests forwarded by the Workstation String Manager Components dealing with creation and maintenance of shared logical strings and user privileges. As shown in the RMS String Manager Component Context Diagram

in Figure 3.2.1-1, the RTS String Manager does not have an interface with the FOS User Interface Subsystem. All user requests and responses are filtered through the Workstation String Manager Component.

Upon request for a logical string to monitor a real-time EOS spacecraft contact, the RMS String Manager Component will create a full complement of software components to satisfy that request. The software components that are created and configured in this scenario include: a real-time Telemetry subsystem, a real-time Command subsystem, a Real-Time Contact Management subsystem, a Ground Script Controller process, a number of DMS Archive processes, and a Parameter Server process. In the event of a request for a backup logical string, the real-time subsystem processes will need to be synchronized with a full complement of similar processes on another Real-Time Server. While it is the responsibility of the Workstation String Manager Component to coordinate this activity, it is the responsibility of the RTS String Manager processes to respond to a snapshot request and load messages to affect this synchronization process. The identical snapshot request and load scenario is performed for backup logical string creation and mirrored string connection. This scenario is explained in detail in Sections 3.2.4.5 and 3.2.4.6 describing mirrored logical string connection.

Communicating with the RMS Resource Monitor Component is unique to the RTS String Manager Component because the Resource Monitor Component is not executed on the User Workstations. The Resource Monitor Component is responsible for mission critical component monitoring. It should be reiterated that there is no mission critical real-time processing on User Workstation within the EOC or ISTs outside of the EOC. Only Real-Time Servers act as host to mission critical real-time processing. For this reason, Resource Monitor Components are installed and execute only on Real-Time Servers within the EOC.

Both the Workstation and Real-Time Server String Manager Components use the CSMS/CSS Directory Naming Service for host information and therefore, a CSMS interface is illustrated on both sides of the Context Diagram in Figure 3.2.1-1.

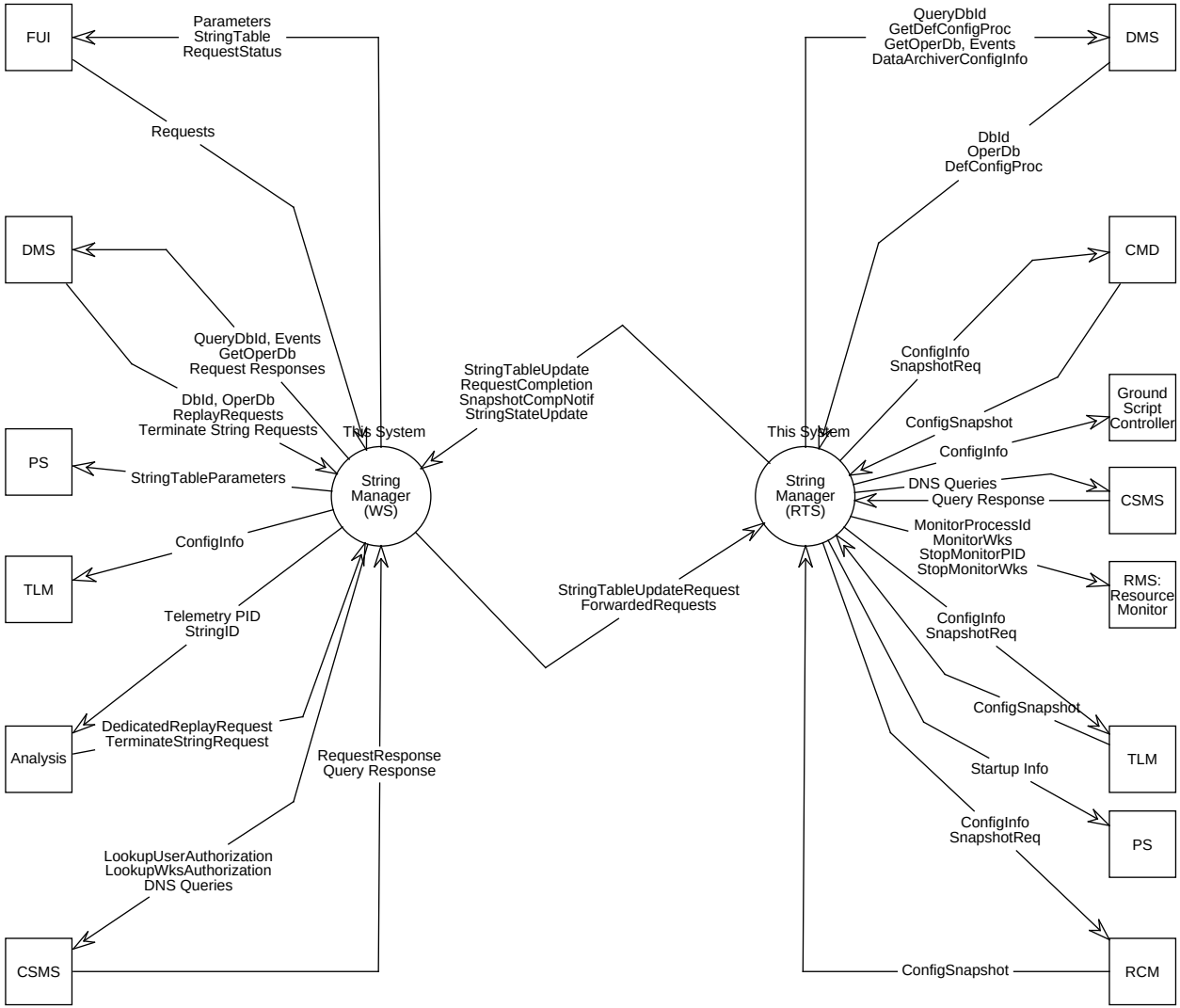


Figure 3.2.1-1. RMS String Manager Component Context Diagram

3.2.2 RMS String Manager Component Interfaces

Table 3.2.2 RMS String Manager Component Interfaces (1 of 3)

Interface Service	Interface Class	Interface Class Description	Service Provider	Service User	Frequency
Configuration of Data Archiver	FdDfRms Config Proxy	Enables RMS to configure the data archiver	Data Archiver	RMS	0-4 per string creation ~ 8 per shift
Configuration of TLM Dump process	FtTl Dump Config	Enables RMS to configure a TLM dump process	TLM	RMS	1 per RT string creation ~ 2 per shift
Configuration of TLM Decomm process	FtTl Telemetry Config	Enables RMS to configure a TLM decomm process	TLM	RMS	1 per TLM type per string creation or connection ~ 12 - 35 per shift
Sending config. to GSC	FuCcGsc Proxy	Enables RMS to configure the GSC	GSC	RMS	1 per RT or SIM string creation ~ 2 per shift
Configuration of CMD Format process	FoGnRms Format Proxy	Enables RMS to configure the CMD Format process	CMD	RMS	1 per RT or SIM string creation ~ 2 per shift
Configuration of CMD Fop process	FoGnCmdRmsFop Proxy	Enables RMS to configure the CMD Fop process	CMD	RMS	1 per RT or SIM string creation ~ 2 per shift
Configuration of CMD Transmit process	FoGnRms Transmit Proxy	Enables RMS to configure the CMD Transmit process	CMD	RMS	1 per RT or SIM string creation ~ 2 per shift
Configuration of RCM process	FoGnRms RcmProxy	Enables RMS to configure an RCM process	RCM	RMS	1 per RT or SIM string creation ~ 2 per shift
Reads requests sent by the RMS String Mgr.	FrGrStrmanResMon IF	Enables the RMS Resource Monitor to read requests from String Mgr.	RMS Resource Monitor	RMS String Mgr.	30 - 100 per contact

Table 3.2.2 RMS String Manager Component Interfaces (2 of 3)

Interface Service	Interface Class	Interface Class Description	Service Provider	Service User	Frequency
Receive WS or RTS RMS requests	FrGrRms WsRmsIF	Enables the respective RMS to receive requests	WS/RTS RMS	WS/RTS RMS	30 -100 per contact
Receives requests from FUI, DMS or ANA	FrGr Request Handler	Enables RMS to receive requests	RMS	FUI ANA DMS	1 - 10 per shift
Authorization and Name Service Queries	FoGnCsms IF	Enables RMS to interface with CSMS name service authorization	CSMS	RMS	1 per service or connection request
Gets and Sets parameters from/to PS	FoPsClient IF	Enables RMS to interface with PS	PS	RMS	30 -100 per contact
Accessing files in DMS	FoDsFile Accessor	Enables RMS to access files within DMS	DMS	RMS	1 - 5 per week
Send replay requests to RMS	FrGrReplayRequest Proxy	Enables external Subsystems to send replay requests	RMS	DMS FUI ANA	1 - 10 per shift
Send requests to RMS	FrGrRms FuiRequest Proxy	Enables FUI to send all types of requests	RMS	FUI	30 - 100 per contact
RTS RMS to WS RMS Interface	FrGrRts Rms Request Proxy	Enables the RTS RMS to send requests to WS RMS	WS RMS	RTS RMS	30 - 100 per contact
RTS RMS to Resource Monitor Interface	FrGrStr ManRes MonProxy	Enables the RTS RMS to send requests to Resource Monitor	Resource Monitor	RTS RMS	30 - 100 per contact

Table 3.2.2 RMS String Manager Component Interfaces (3 of 3)

Interface Service	Interface Class	Interface Class Description	Service Provider	Service User	Frequency
Sends event messages to event logger	FdEvEventLogger	Enables RMS to log event messages	DMS	RMS	30 - 100 per contact
WS RMS to RTS RMS Interface	FrGrWs Rms Request Proxy	Enables the WS RMS to send requests to the RTS RMS	RTS RMS	WS RMS	30 - 100 per contact

3.2.3 RMS String Manager Component Object Model

Figure 3.2.3-1 illustrates a top level view of the RMS String Manager Component. Subsequent Figures illustrate, in more detail, specific objects as noted on Figure 3.2.3-1. The objects shown on Figure 3.2.3-1 allow the RMS to create, configure, and reconfigure string software resources.

Two types of Rogue Wave objects are utilized to facilitate the String Manager in performing its tasks. These include the RWSet object and the RWCString object. The RWSet object is a collection class that can store other String Manager objects. The RWCString object is used to manage character sets.

There are several "proxy" and "receiver classes" shown on the object model to facilitate communication with other RMS processes. The FrGrWsRmsRequestProxy allows a WS RMS to send Request objects to the RTS RMS. This proxy class will receive a status on each request it sends to a RTS RMS. For every RTS that is used, there will be a FrGrWsRmsRequestProxy that is stored in a RWSet. Each RTS RMS contains a FrGrRtsRmsRequestProxy that allows the RTS RMS to send Request objects to the WS RMS. Like the FrGrWsRmsRequestProxy, the FrGrRtsRmsRequestProxy will receive a status on each request it sends to a RTS RMS. The FrGrRtsRmsRequestProxy provides the capability to multicast objects to every WS RMS when necessary. This will mainly be used to multicast the RTS RMS String Table. Both the WS RMS and the RTS RMS will have a FrGrRmsWsRmsIF object for receiving objects from either a FrGrWsRmsRequestProxy or a FrGrRtsRmsRequestProxy. When an object is received, it will place it in a Request Queue. The queue will be implemented with a RWSet. The FrGrController will call the FrGrRmsWsRmsIF object's CheckQueue operation to retrieve an object from the queue. The FrGrStrManResMonProxy class will allow the String Manager to send Requests to the RMS Resource Monitor.

Other proxy and receiver classes shown on the object model allow the String Manager to communicate with external subsystems. The FoPsClientIF object enables the String Manager to provide string table parameters. The FoGnCsmsIF objects enables the String Manager to utilize the CSMS name server as well as utilize CSMS security operations. Security operations could include ensuring whether a user is permitted to have command or ground control authority. The FdEvEventLogger object enables the String Manager to send events to DMS. The FoDsFileAccessor enables the String Manager to access files within DMS. The FuCcGscProxy object enables the String Manager to send requests to the Ground Script Controller process. The

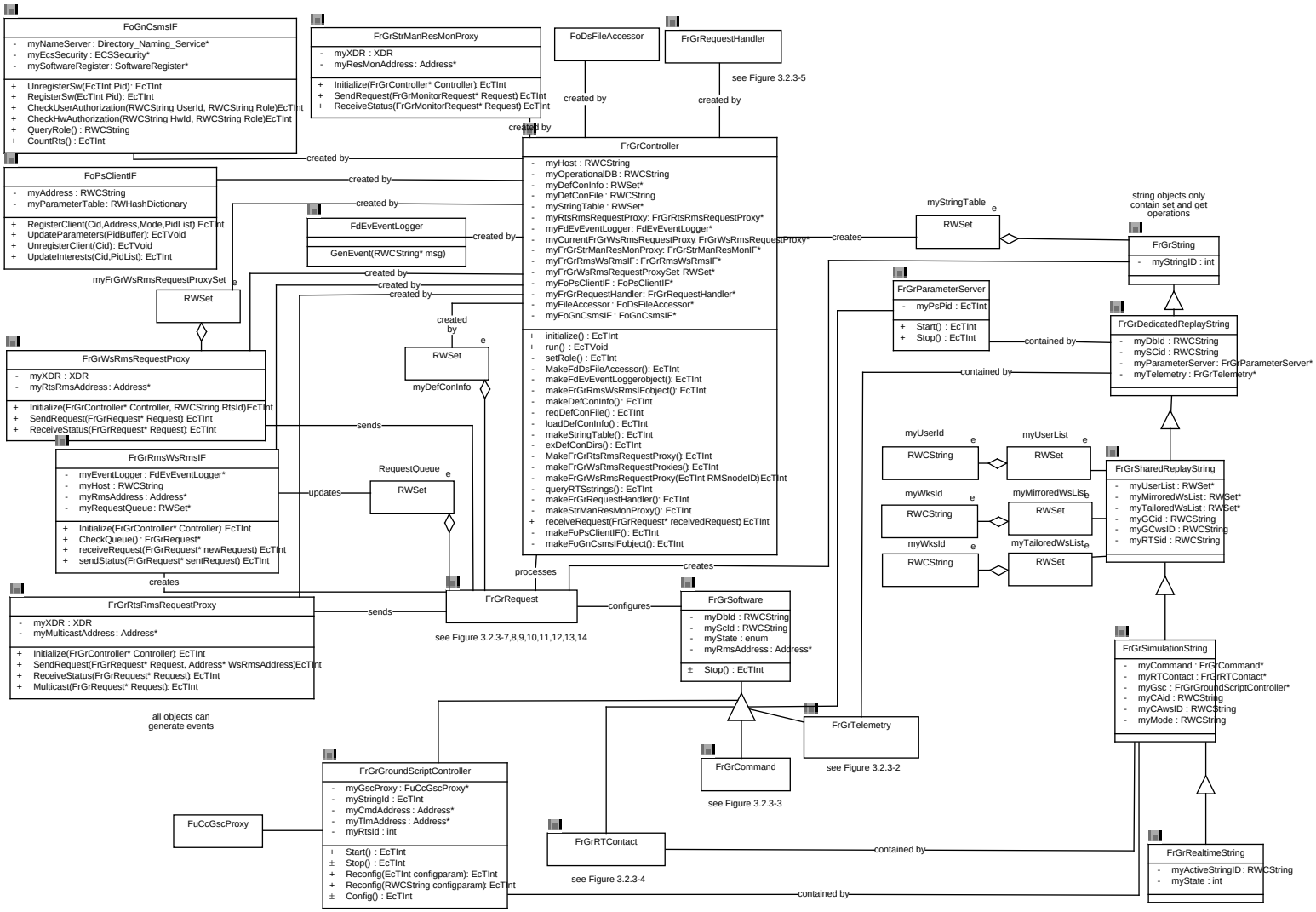


Figure 3.2.3-1. RMS String Manager Component Object Model

FrGrRequestHandler object is a receiver object. This object receives objects from FUI, DMS, and Analysis. These objects are shown in Figure 3.2.3-6. Figure 3.2.3-5 illustrates the FrGrRequestHandler object in much more detail.

There are several string objects shown on the diagram. These objects contain string related information as well as pointers to String Manager software resource objects. String object operations are limited to "set" and "get" operations. The FrGrString object is an abstract class that only contains a String ID. The FrGrDedicatedReplayString is created when a user wishes to create telemetry processes on a workstation with a configuration dedicated solely to the user. The configuration would not be shared by other users. The FrGrSharedReplayString is created when a user creates telemetry processes on a RTS with a configuration that is shared by additional users. These users are tracked by the corresponding User and Workstation Lists associated with the object. The FrGrSimulationString is created when a user wishes to create a simulation string. This string will have similar attributes to a FrGrRealtimeString. The FrGrRealtimeString will contain additional attributes for identifying it as a backup or active string. If the FrGrRealtimeString is a backup string, it identifies the string that it is backing up. The FrGrRealtimeString is the only string that will be backed up or failed over by the String Manager. When a string object is created, it is placed in a string table. The string table is implemented with a RWSet collection class.

Many of the objects shown on the diagram will be used to identify string resources. The FrGrSoftware object is an abstract object. It contains attributes and a Stop operation that is inherited by other string resource objects. The FrGrGroundScriptController object is used to start, stop, configure, and modify the configuration of the FUI Ground Script Controller process. The FrGrRTContact object is used to start, stop, configure, and modify the configuration of the RCM processes. In addition, it enables the String Manager to take a configuration snapshot of the RCM processes. This configuration snapshot is used to configure backup RCM processes. The FrGrRTContact object is shown in more detail in Figure 3.2.3-4. The FrGrCommand object is used to start, stop, configure, and modify the configuration of the Command processes. In addition, it enables the String Manager to take a configuration snapshot of the Command processes. This configuration snapshot is used to configure backup Command processes. The FrGrCommand object is shown in more detail in Figure 3.2.3-3. The FrGrTelemetry object is used to start, stop, configure, and modify the configuration of the Telemetry processes. In addition, it enables the String Manager to take a configuration snapshot of the Telemetry processes. This configuration snapshot is used to configure backup or mirrored Telemetry processes. The FrGrTelemetry object is shown in more detail in Figure 3.2.3-2. The FrGrParameterServer object is used to start and stop the Parameter Server process.

The FrGrRequest object is an abstract class that contains a virtual Execute operation. Several Request objects are derived from the FrGrRequest object and each will overwrite the Execute operation. The Execute operation is called by the FrGrController object and is responsible for containing all functionality or calling any subroutines necessary for processing a particular request. The classes derived from the FrGrRequest object are shown in Figures 3.2.3-7,8,9,10,11,12,13,14.

The FrGrController object enables the String Manager to initialize itself, communicate with other processes, and initiate the processing of requests. At initialization, the FrGrController determines what type of host it is running on via the CSMS name server. A host can be a real-time server (RTS) or a workstation (WS). Once the host is determined, it creates appropriate proxy objects, receiver objects, and collection classes. If the Controller is running on a RTS, it creates default

configuration requests from information in the default configuration file that is retrieved from DMS. These requests are notified by the Controller to execute. Default strings are created as a result of processing the default configuration requests. Once the Controller has initialized, it enters a "run" state. In this state, the Controller will notify requests to execute as they are received from other processes.

The FrGrTelemetry object is derived from the FrGrSoftware object. It contains a FrGrTelemetryProcess object for every type of telemetry process that it needs to communicate with. For a Real-Time Operational String, the FrGrTelemetry object would point to a full range of FrGrTelemetryProcess objects. This would include a state check process, a dump process, and three decommutation processes that will be used for decommutating housekeeping, health&safety, and standby telemetry. The FrGrTelemetry object will notify the FrGrTelemetryProcess object to start, stop, request a configuration snapshot, modify an existing configuration, configure using a snapshot configuration file, or configure using a database. Each FrGrTelemetryProcess object is responsible for communicating with a single telemetry process via a FtTlTelemetryConfig object or a FtTlDumpConfig object. The decommutation and dump processes require a DMS Data Archiver process to be configured in order for the incoming telemetry data to be archived. If the FrGrTelemetryProcess is associated with a decommutation or dump process on the RTS, the FrGrDataArchiver object is created. This object is responsible for starting, stopping, and configuring the data archive process. The FrGrDataArchiver object communicates with the Data Archiver process via the FdCfRMSConfigProxy.

The FrGrCommand object is derived from the FrGrSoftware object. It contains a FrGrCommandProcess object for every type of command process that it needs to communicate with. This would include a Format process, a FOP process, and an Transmit process. The FrGrCommand object will notify the FrGrCommandProcess object to start, stop, request a configuration snapshot, modify an existing configuration, configure using a snapshot configuration file, or configure using a database. Each FrGrCommandProcess object is responsible for communicating with a single command process via a FoGnRmsFormatProxy object, a FoGnCmdFopRmsProxy object, or a FoGnRmsTransmitProxy object.

The FrGrRTContact object is derived from the FrGrSoftware object. It contains a FrGrRcmProcess object for every type of Real-time Contact Management process that it needs to communicate with. This would include a NoutMgr process and an EoutMgr process. The RCM subsystem consists of two additional processes. These are the NinMgr process and the EinMgr process. However, the String Manager will send information to and start these processes via the NoutMgr and EoutMgr processes. The FrGrRTContact object will notify the FrGrRcmProcess object to start, stop, request a configuration snapshot, modify an existing configuration, configure using a snapshot configuration file, or configure using a database. Each FrGrRcmProcess object is responsible for communicating with a single RCM process via a FoGnRmsRcmProxy object. The NinMgr and EinMgr processes require a DMS Data Archiver process to be configured in order for the incoming Nascom blocks and Customer Operations Data Accounting (CODA) reports to be archived. The NoutMgr and EoutMgr processes will be notified of the Data Archiver address when they are started. They will pass this information to the NinMgr and EinMgr processes. In order to create and configure the Data Archiver process, a FrGrDataArchiver object is created by the FrGrRcmProcess object. This object is responsible for starting, stopping, and configuring a data archive process. The FrGrDataArchiver object communicates with the Data Archiver process via the FdCfRMSConfigProxy.

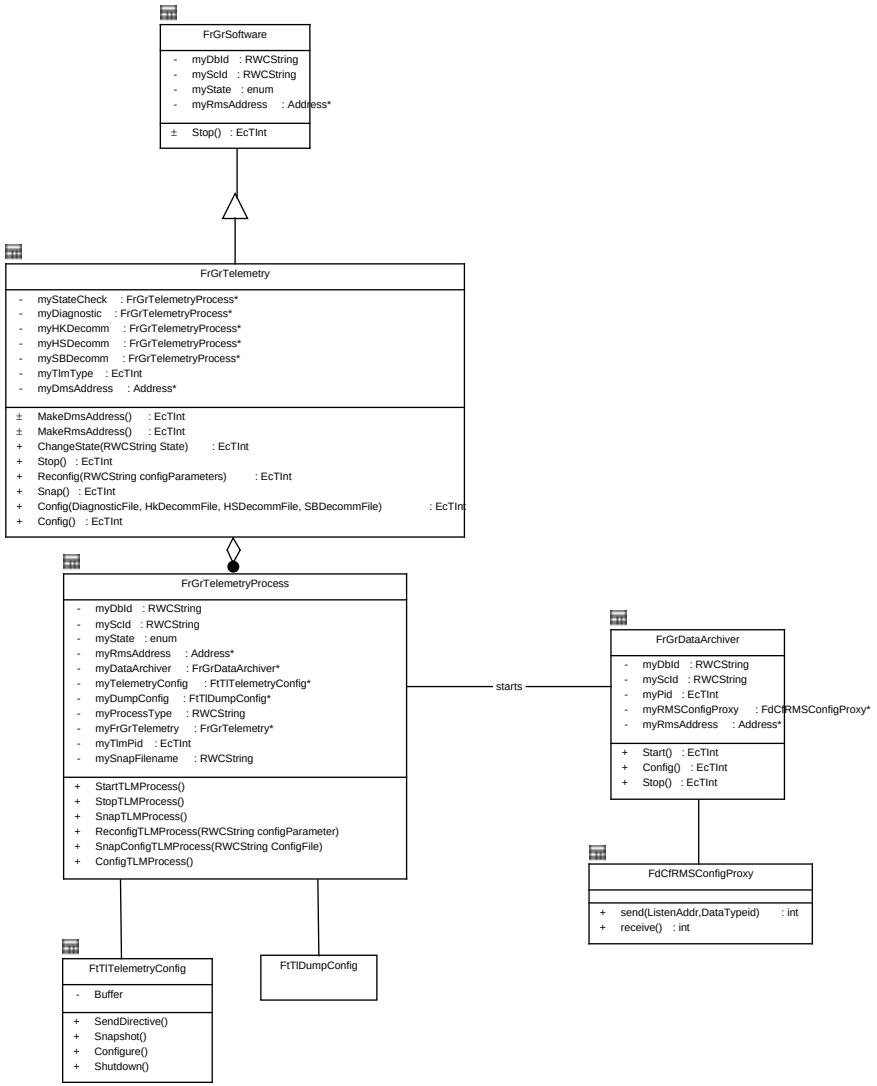


Figure 3.2.3-2. RMS String Manager Component FrGrTelemetry Object Model

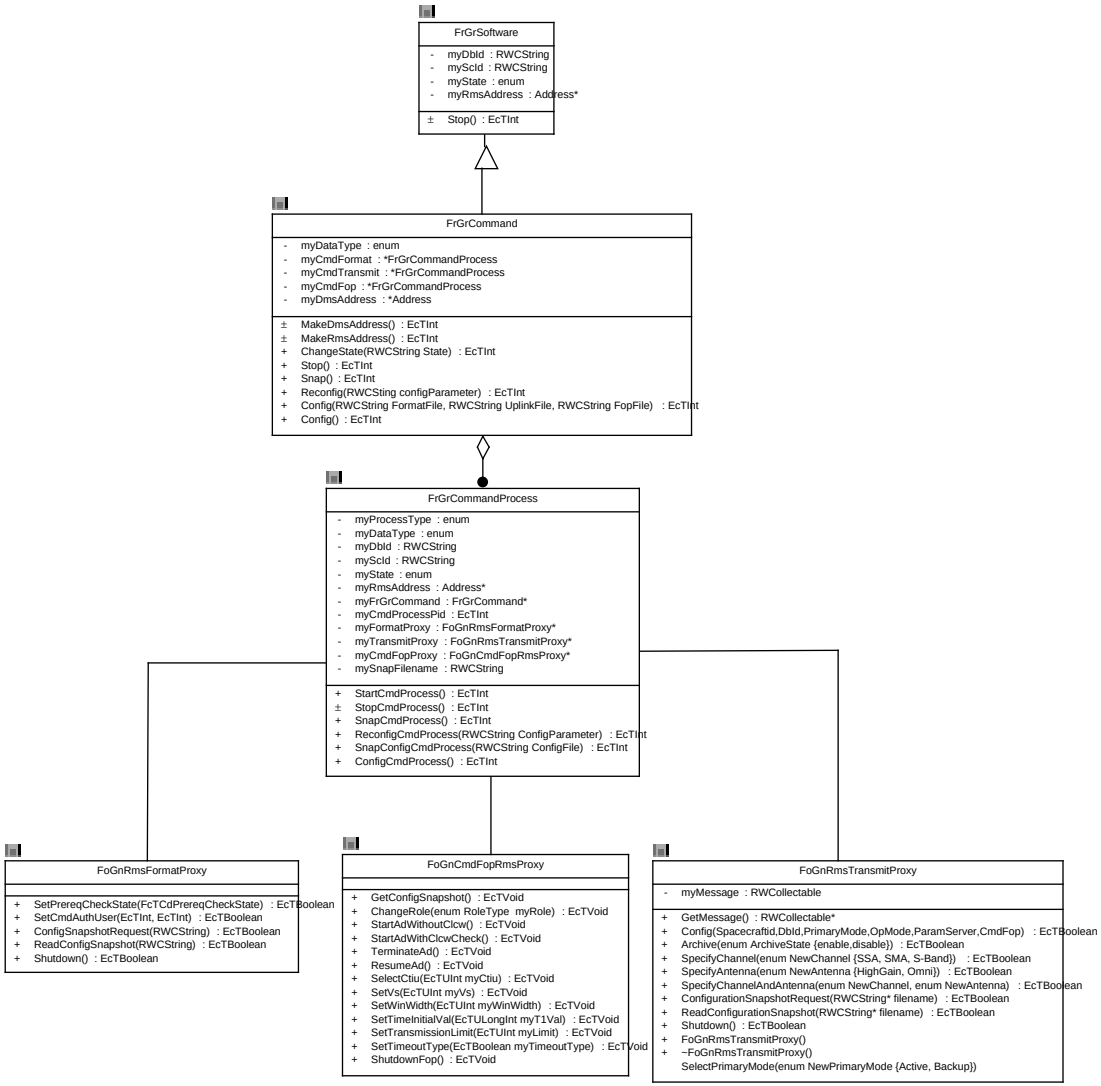


Figure 3.2.3-3. RMS String Manager Component FrGrCommand Object Model

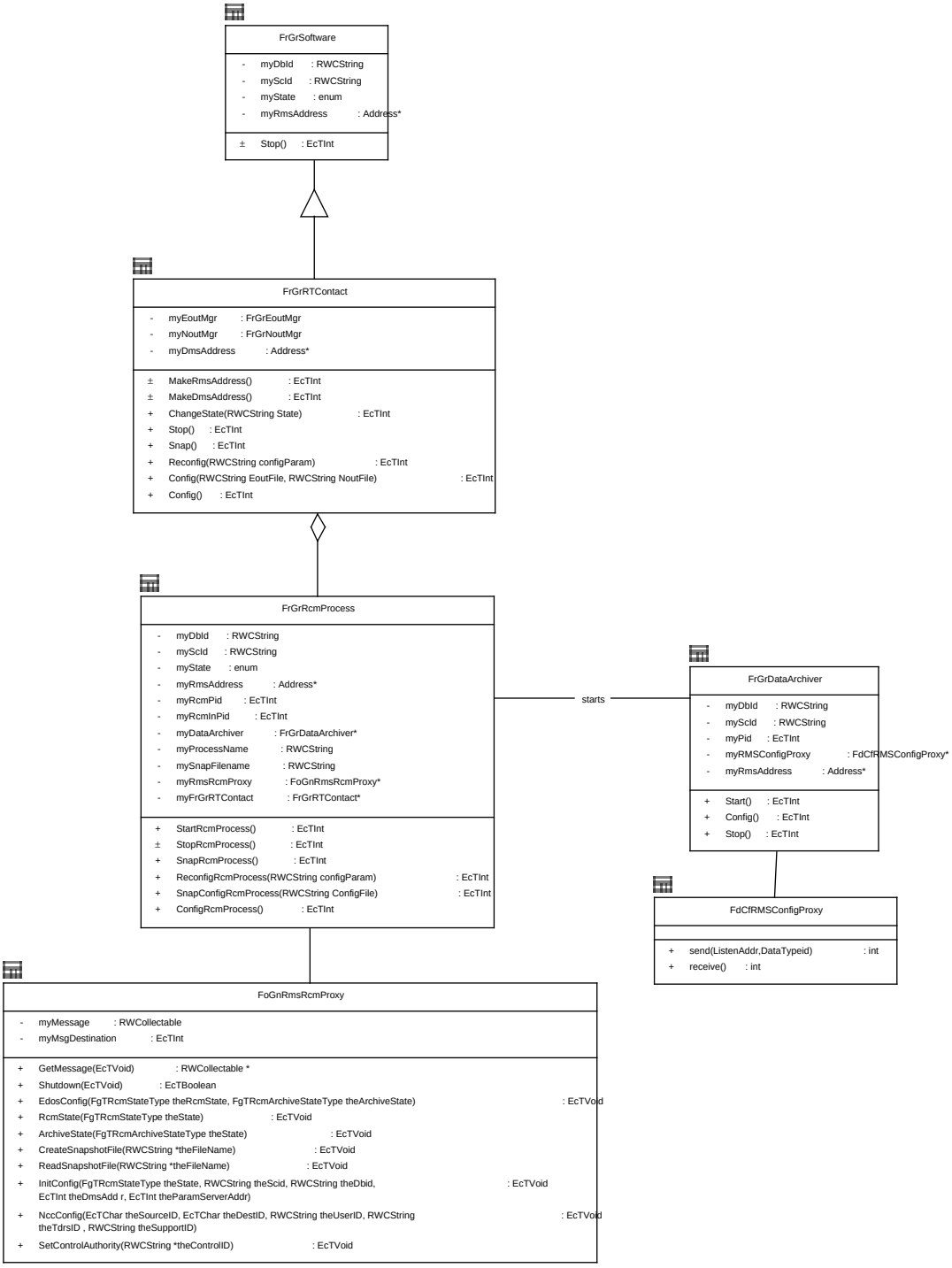


Figure 3.2.3-4. RMS String Manager Component FrGrRTContact Object Model

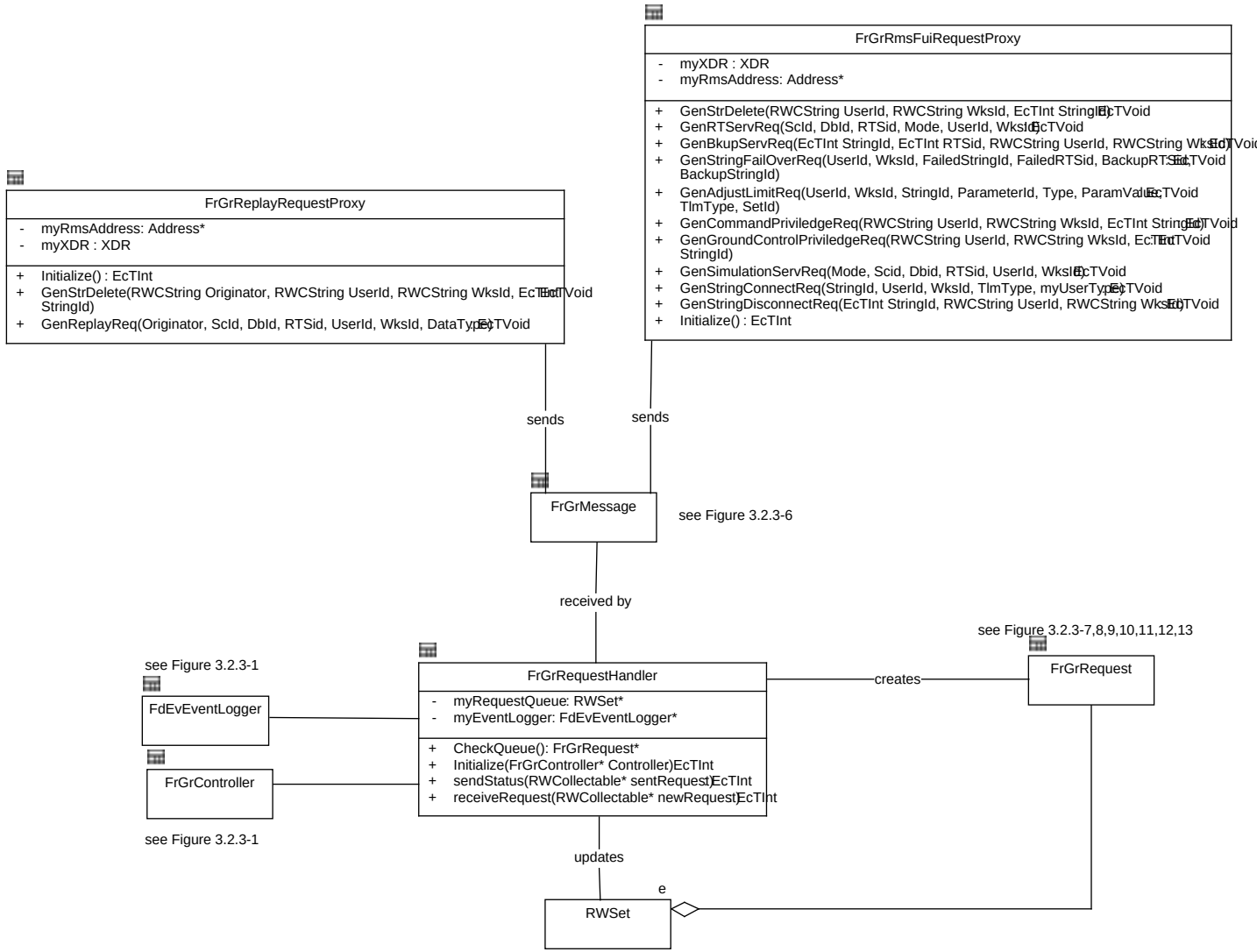


Figure 3.2.3-5. RMS String Manager Component FrGrRequestHandler Object Model

The FrGrRequestHandler object is responsible for receiving a Message object from FUI, DMS, or the Analysis Subsystem, instantiating the appropriate Request object, placing the Request object in a queue, retrieving a Request object from the queue, and returning a status when the request has been processed. If necessary, it can generate an event via the FdEvEventLogger object. The FrGrReplayRequestProxy object will reside in a service user's process. For clarity, it is placed in this object model. This proxy object can send a Replay Service Request and a String Delete Request. The FrGrRmsFuiRequestProxy object can send requests to create strings, modify string configurations, failover strings, delete strings, connect to strings, disconnect from strings, and take the Ground Control or Command Authority privilege on a particular string. Figure 3.2.3-6 illustrates the Message objects that the FrGrReplayRequestProxy and the FrGrRmsFuiRequestProxy objects will send. When the FrGrRequestHandler object receives the Message object, it instantiates the appropriate Request object and places it in a queue. The queue is implemented via a RWSet. When requested by the FrGrController object, a Request object is retrieved from the queue and returned to the Controller. Figures 3.2.3-7,8,9,10,11,12,13 illustrate, in more detail, the Request objects.

The FrGrMessage object is derived off of the RWCollectable object. This is necessary in order to inherit needed operations for flattening objects to a stream. When an object is passed between processes, it is flattened to a stream. FrGrMessage, FrGrStringAccessMessage, and FrGrStringCreateMessage are abstract classes containing attributes that derived objects will inherit. When the FrGrRequestHandler receives any of the other derived objects, it instantiates a particular Request object. These Request objects are shown in Figures 3.2.3-7,8,9,10,11,12,13. For example, when the FrGrRequestHandler receives the FrGrStringFailoverReqMessage, it instantiates a FrGrStringFailoverRequest object and places it in a queue. The only operations that the Message objects contain are "set" and "get" operations.

The FrGrStringAccessRequest object is derived off of the FrGrRequest object. FrGrRequest is an abstract class that contains attributes and an Execute operation that derived classes will inherit. FrGrStringAccessRequest contains additional attributes and a FindString operation that derived objects will inherit. The FrGrStringConnectRequest object contains attributes and operations necessary for connecting a user to an existing string. It will create telemetry processes on a workstation for telemetry decommutation. The FrGrStringDisconnectRequest contains attributes and operations necessary for disconnecting a user from a string. It will delete telemetry processes on a workstation. The FrGrSnapshotCompNotif object is utilized when a configuration snapshot is retrieved from RTS telemetry processes. When a user connects, the FrGrStringConnectRequest object is forwarded from the WS RMS to the RTS RMS. The RTS RMS requests a configuration snapshot from a telemetry process and the configuration snapshot information is written to a file on the requesting workstation. The RTS RMS returns with the FrGrSnapshotCompNotif object. The FrGrStringConnectRequest notifies the FrGrSnapshotCompNotif object to execute. The FrGrSnapshotCompNotif object will create and configure the telemetry processes using the configuration snapshot file.

The FrGrBackupServiceRequest object is derived off of the FrGrRequest object. FrGrRequest is an abstract class that contains attributes and an Execute operation that derived classes will inherit. The FrGrBackupServiceRequest object contains attributes and operations necessary for creating a backup Real-Time Operational String. It will request configuration snapshots from active real-time processes and configure a Real-Time Operational String using the configuration snapshot files that were created. A backup Real-Time Operational String is necessary for failing over an active string when a failure occurs.

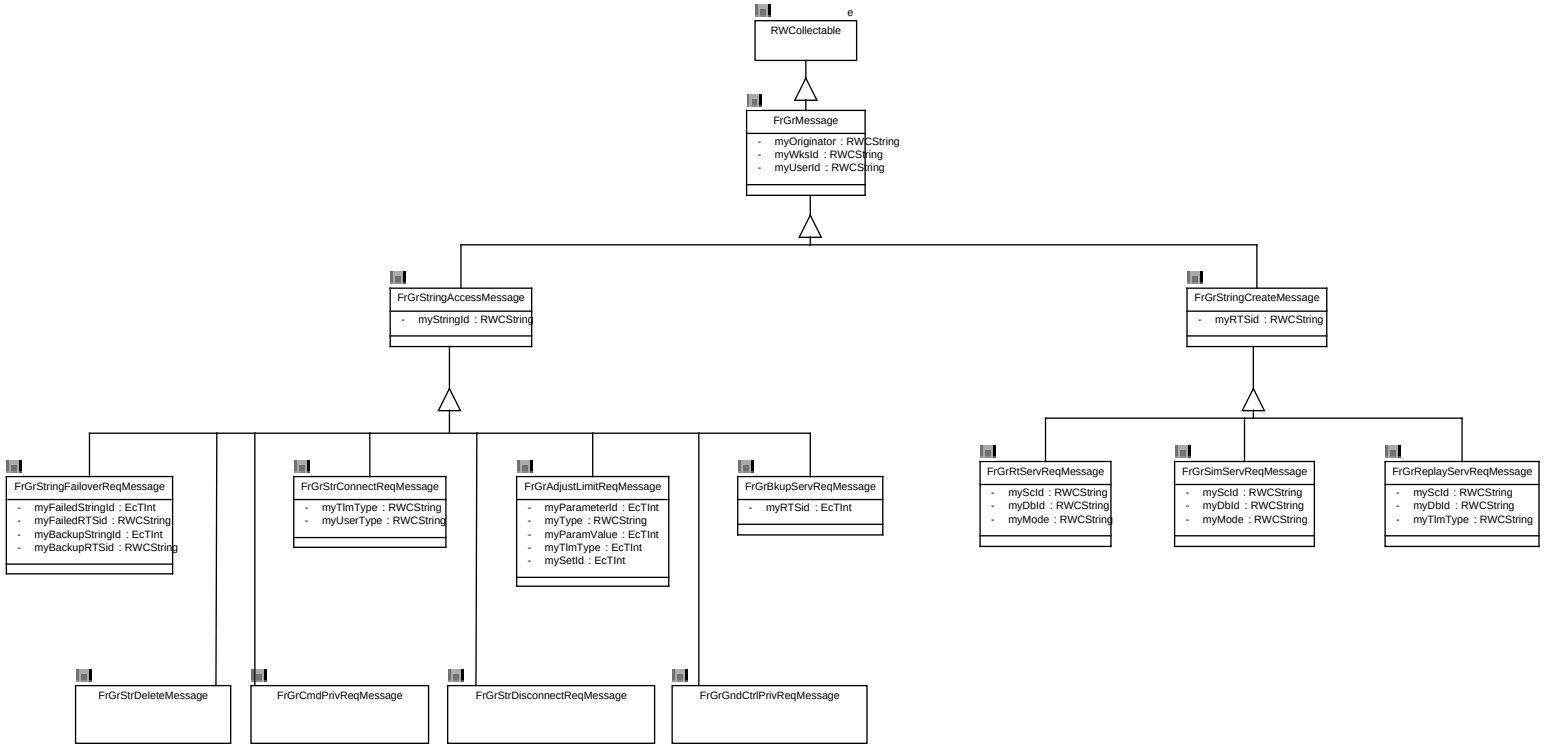


Figure 3.2.3-6. RMS String Manager Component FrGrMessage Object Model

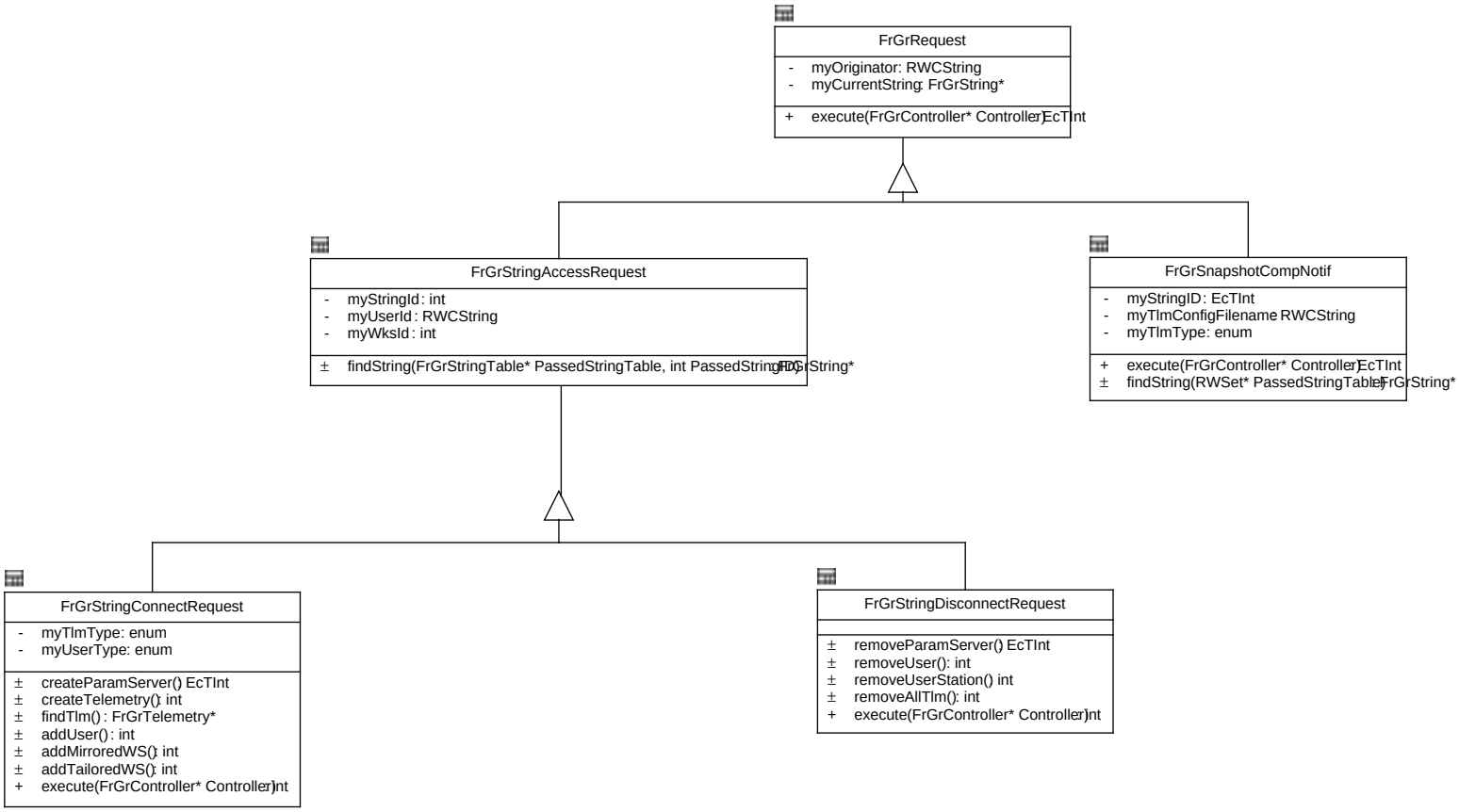


Figure 3.2.3-7. RMS String Manager Component FrGrStringAccessRequest Object

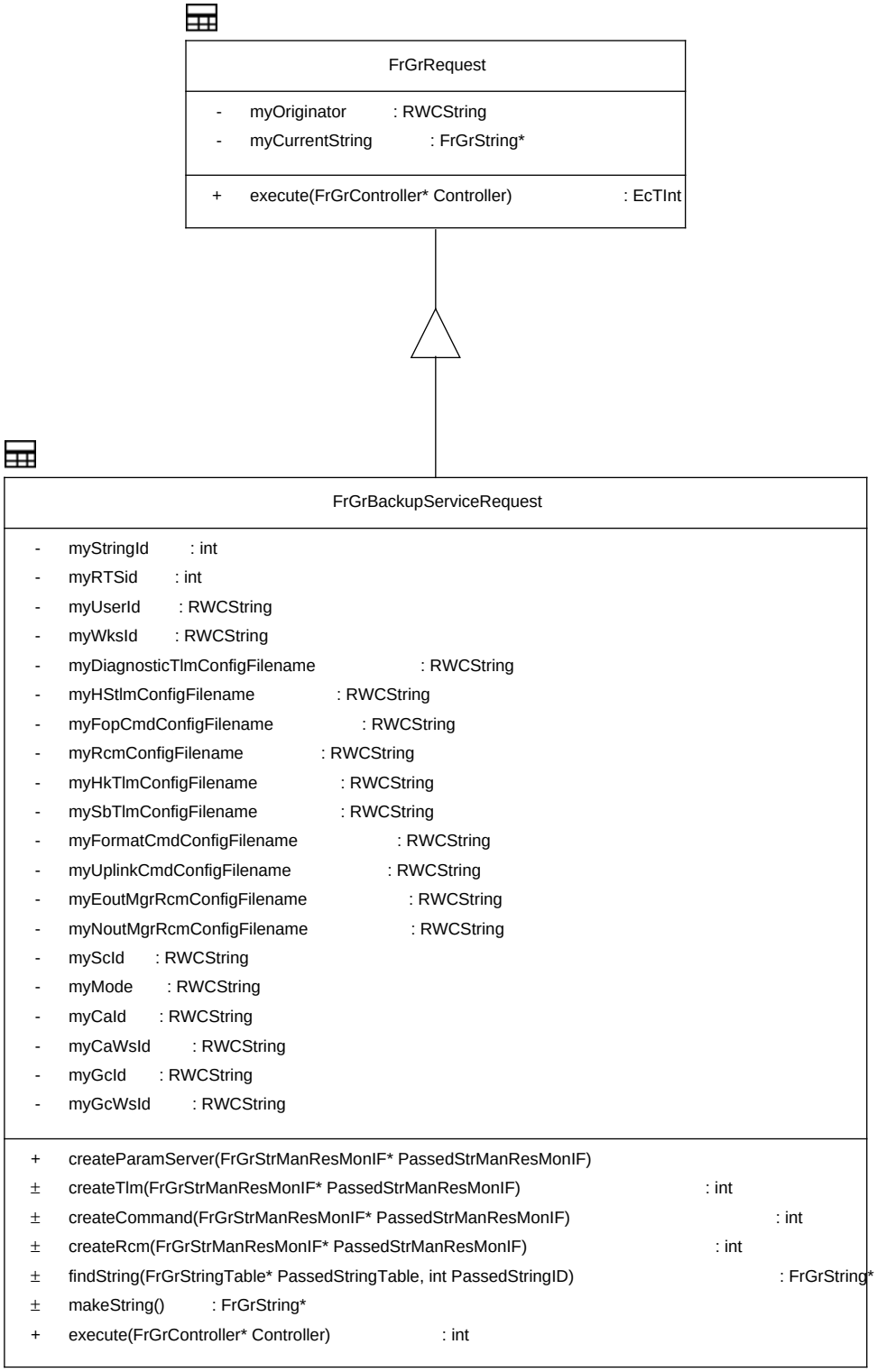


Figure 3.2.3-8. RMS String Manager Component FrGrBackupServiceRequest Object Model

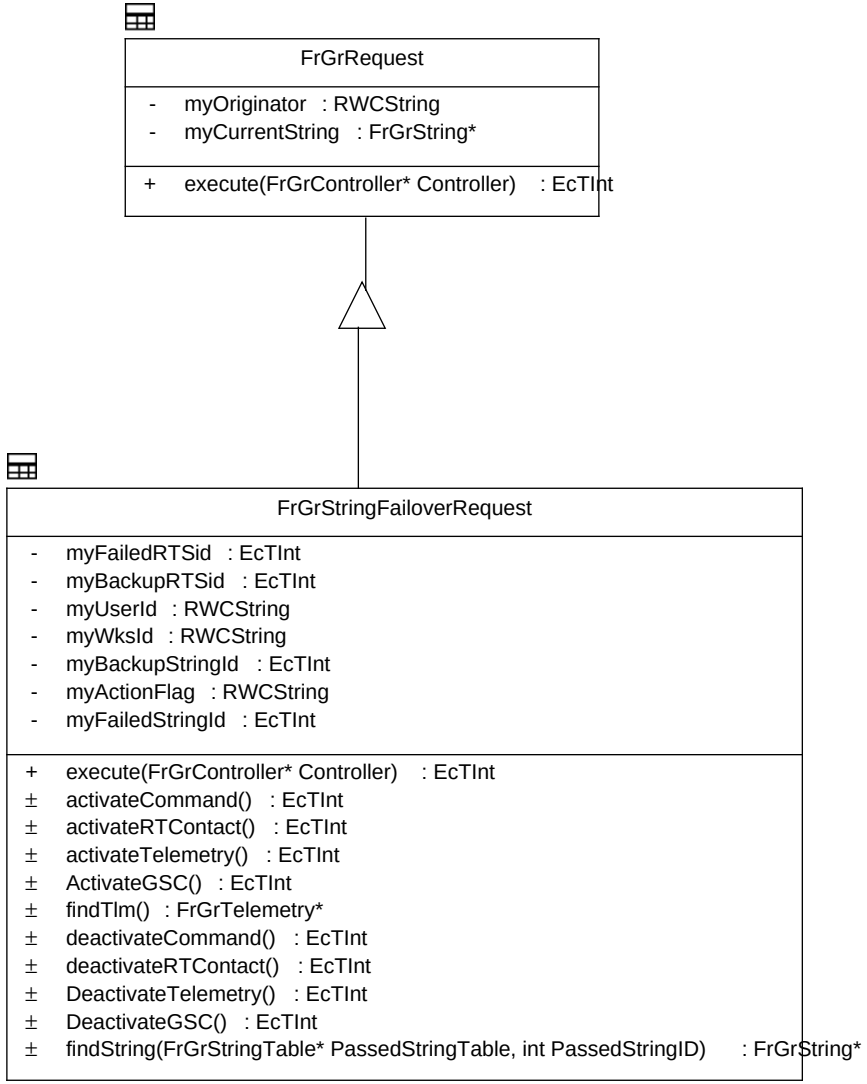


Figure 3.2.3-9. RMS String Manager Component FrGrStringFailoverRequest Object Model

The FrGrStringFailoverRequest object is derived off of the FrGrRequest object. FrGrRequest is an abstract class that contains attributes and an Execute operation that derived classes will inherit. The FrGrStringFailoverRequest object contains attributes and operations necessary for deactivating a failed Real-Time Operational String and activating a backup Real-Time Operational String.

The FrGrGroundControlRequest object is derived off of the FrGrRequest object. FrGrRequest is an abstract class that contains attributes and an Execute operation that derived classes will inherit. The FrGrGroundControlRequest is an abstract class that Configuration Change Requests will be derived from. The FrGrAdjustLimitRequest is an example of a single Configuration Change Request. The FrGrAdjustLimitRequest object contains attributes and operations necessary for forwarding a limit adjustment to a telemetry process. If the limit change is for the telemetry on the RTS, any mirrored workstations will receive the FrGrAdjustLimitRequest after the WS RMS forwards the Request to the RTS RMS. All telemetry Configuration Change Requests will be handled in the same way. The information that is sent to the processes will be different. Command, RCM, and the Ground Script Controller will be handled in a similar way. The information forwarded to the processes will be different and after the WS RMS forwards the Configuration Change Request to the RTS RMS, it will not be sent back to any mirrored workstations. This is not necessary since Command, RCM, and Ground Script Controller processes do not reside on the workstation.

The FrGrPrivilegeRequest object is derived off of the FrGrRequest object. FrGrRequest is an abstract class that contains attributes and an Execute operation that derived classes will inherit. The FrGrPrivilegeRequest is an abstract class that the FrGrCommandPrivilegeRequest and the FrGrGroundControlPrivilegeRequest objects are derived from. It contains attributes and operations needed by the FrGrGroundControlPrivilegeRequest and the FrGrCommandPrivilegeRequest. The FrGrCommandPrivilegeRequest object contains attributes and operations necessary for changing command authority on a Real-time or Simulation String. Only a user with command authority is capable of commanding a particular spacecraft. The FrGrGroundControlPrivilegeRequest object contains attributes and operations necessary for changing ground control authority on a Real-time, Simulation, or Shared String. Only a user with ground control authority is capable of making configuration changes to string processes that reside on the RTS.

The FrGrServiceRequest object is derived off of the FrGrRequest object. FrGrRequest is an abstract class that contains attributes and an Execute operation that derived classes will inherit. The FrGrServiceRequest is an abstract class that the FrGrRealtimeServiceRequest, FrGrSimulationServiceRequest, and the FrGrReplayServiceRequest objects are derived from. It contains attributes and a MakeString operation needed by the derived Requests. The FrGrRealtimeServiceRequest object contains all functionality needed for creation of a Real-Time String. The FrGrSimulationServiceRequest object contains all functionality needed for creation of a Simulation String. The FrGrReplayServiceRequest object contains all functionality needed for creation of a Dedicated Replay or Shared Replay String.

The FrGrStringDeleteRequest object is derived off of the FrGrRequest object. FrGrRequest is an abstract class that contains attributes and an Execute operation that derived classes will inherit. The FrGrStringDeleteRequest object contains all functionality needed for deletion of a string and its associated processes.

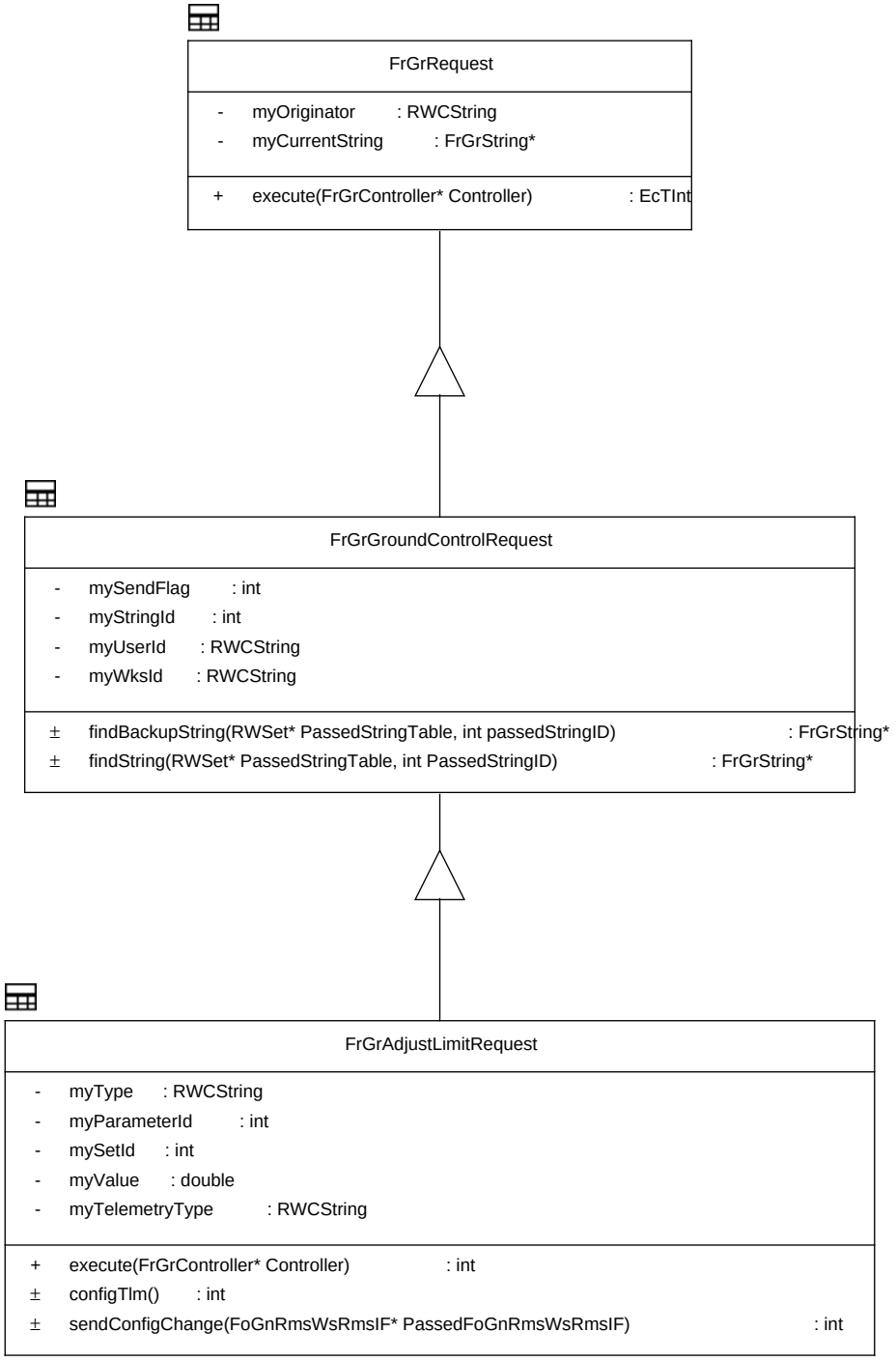


Figure 3.2.3-10. RMS String Manager Component FrGrAdjustLimitRequest Object Model

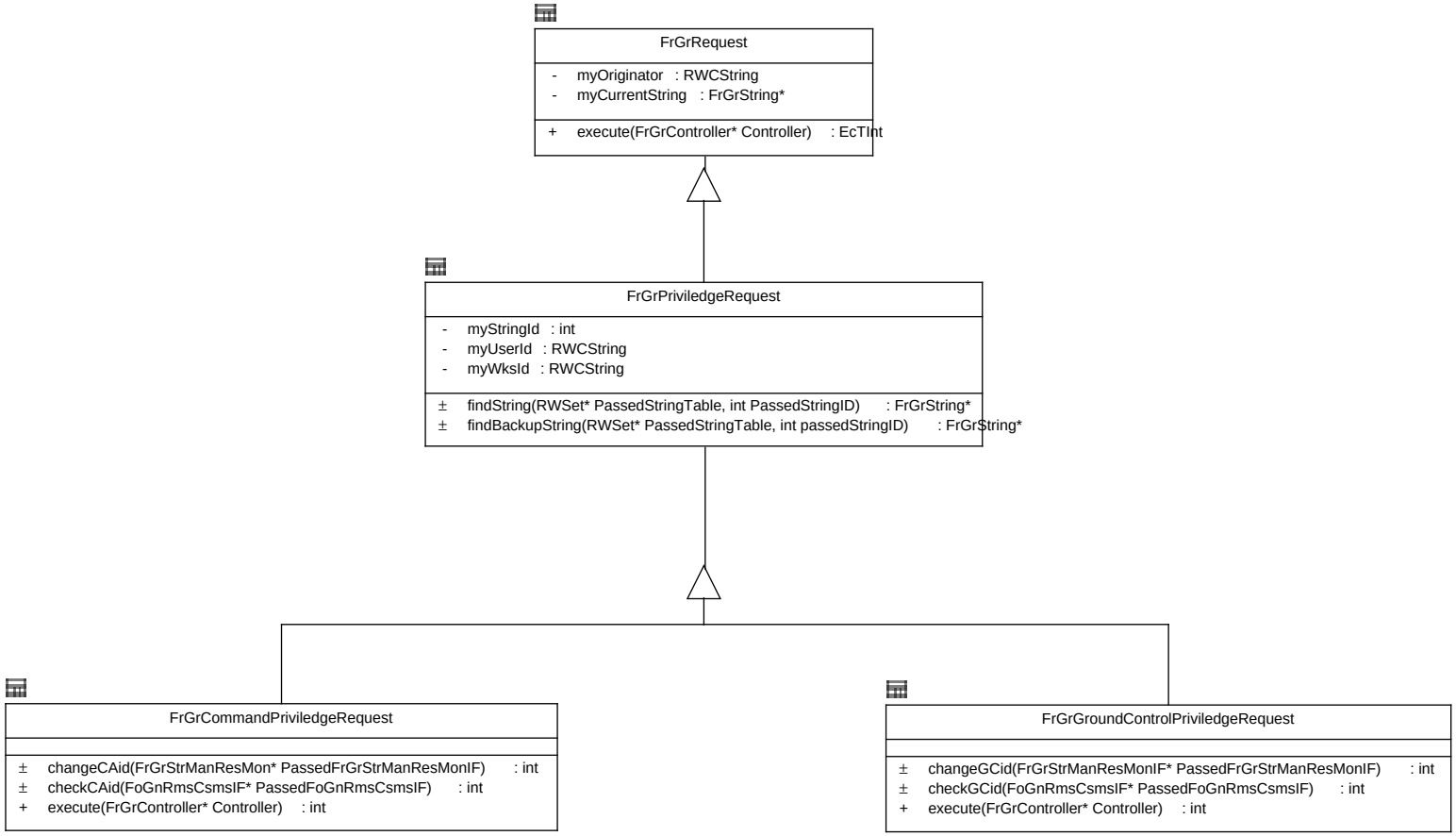


Figure 3.2.3-11. RMS String Manager Component FrGrPrivilegeRequest Object Model

Figure 3.2.3-12. RMS String Manager Component FrGrServiceRequest Object Model

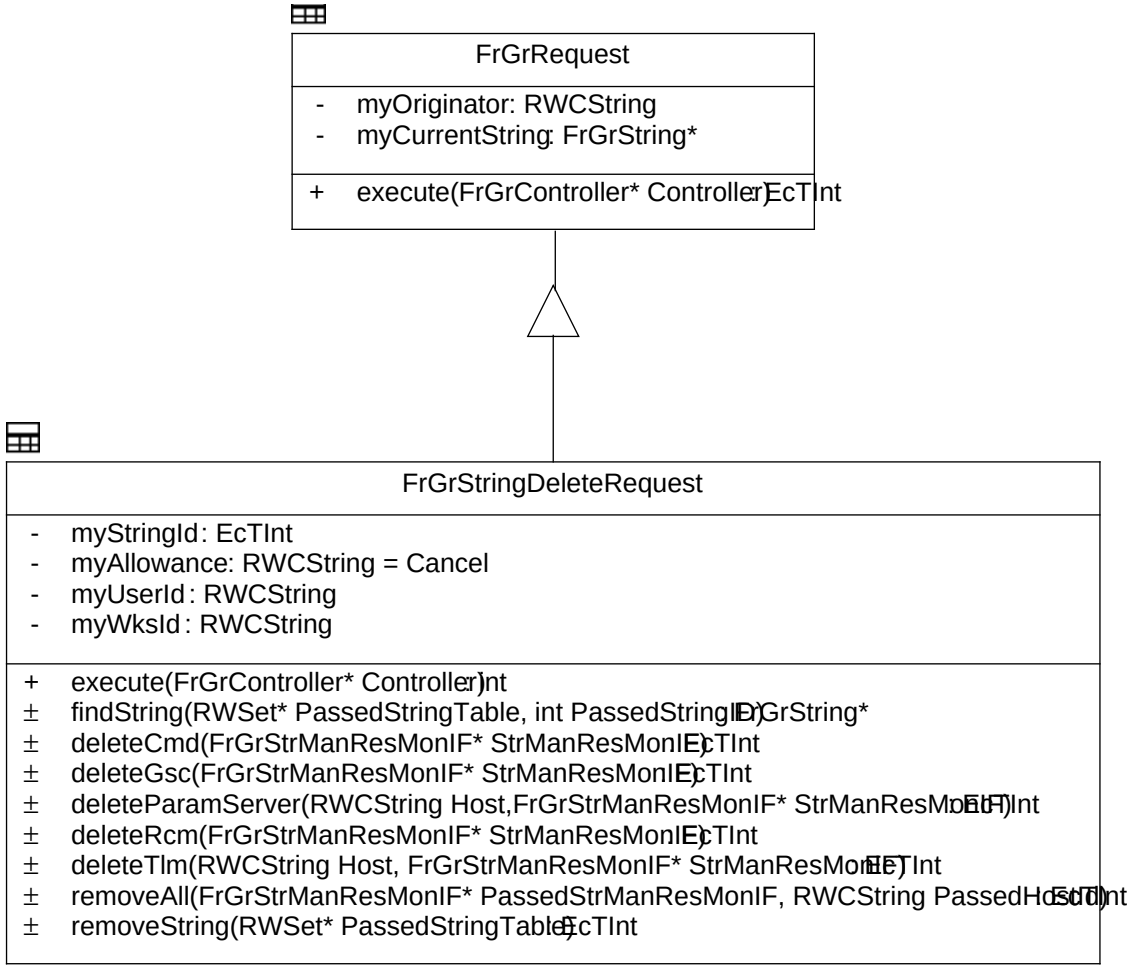


Figure 3.2.3-13. RMS String Manager Component FrGrStringDeleteRequest Object Model

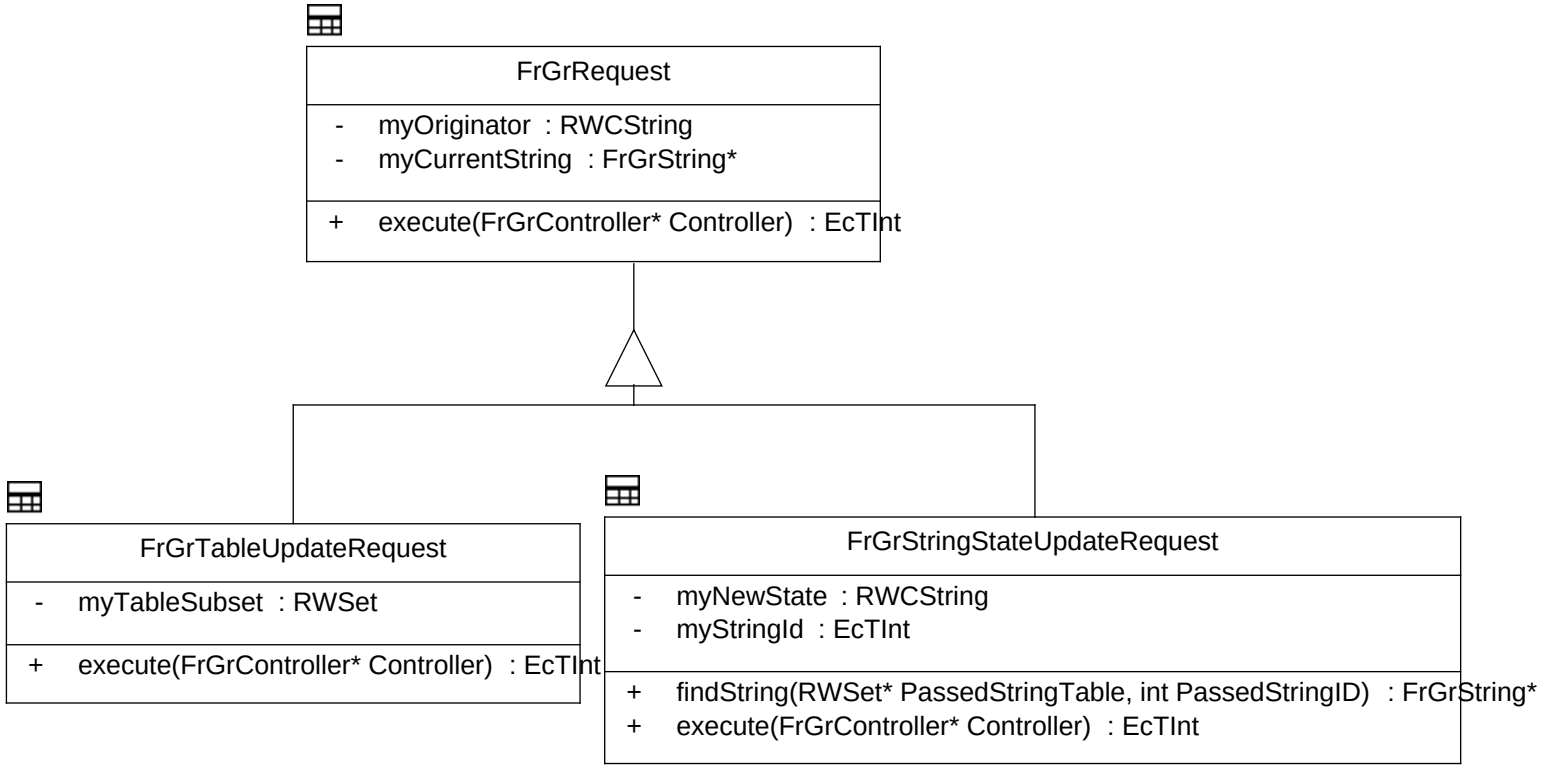


Figure 3.2.3-14. RMS String Manager Component FrGrTableUpdateRequest Object Model

The FrGrTableUpdateRequest and FrGrStringStateUpdateRequest objects are derived off of the FrGrRequest object. FrGrRequest is an abstract class that contains attributes and an Execute operation that derived classes will inherit. The FrGrTableUpdateRequest object contains all functionality needed for updating a String Table with a Table Subset. When a RTS RMS String Table is modified, it is multicasted to every WS RMS via the FrGrTableUpdateRequest. The WS RMS String Table is an inclusive set of all of the RTS RMS String Tables. The WS RMS will receive the FrGrTableUpdateRequest and include the myTableSubset into its WS RMS String Table. When a RTS fails and is incommunicable, its strings are failed over by the user. When the RTS RMS on the backup RTS receives the String Failover Request, it will check the Request to determine if the active RTS is communicable. If not, the RTS RMS will multicast to every WS RMS a FrGrStringStateUpdateRequest. The WS RMS will receive the Request and update its WS RMS String Table by changing the state of the failed RTS String from Active to Inactive.

3.2.4 RMS String Manager Component Dynamic Model

The following are the RMS String Manager Component scenarios which are defined in this section.

- Initialization of the RMS residing on the Workstation
- Initialization of the RMS residing on the Real-Time Server
- Request for a Real-Time Service Arrives on the Workstation
- Request for a Real-Time Service Arrives on the Real-Time Server
- Request for a Mirrored String Connection Arrives on the Workstation
- Request for a Mirrored String Connection Arrives on the Real-Time Server
- Request for Command Authority Arrives on the Workstation
- Request for Command Authority Arrives on the Real-Time Server
- Request for Telemetry Configuration Change Arrives on the Workstation
- Request for Telemetry Configuration Change Arrives on the Real-Time Server
- Request for Dedicated Replay Telemetry Arrives on the Workstation from DMS
- Request for Dedicated Replay Telemetry Arrives on the Workstation from Analysis
- Request for a String Failover Arrives on the Workstation
- Request for String Deactivation Arrives on the Real-Time Server
- Request for String Activation Arrives on the Real-Time Server

3.2.4.1 Initialization of RMS Residing on the Workstation Scenario

3.2.4.1.1 Initialization of RMS Residing on the Workstation Abstract

The purpose of the Initialization of RMS Residing on the Workstation scenario is to describe the process by which the RMS is initialized upon system startup of an EOC User Workstation.

3.2.4.1.2 Initialization of RMS Residing on the Workstation Summary Information

Interfaces:

SCDO/CSS Name Server

Data Management Subsystem

Parameter Server Subsystem

Stimulus:

The RMS software is executed on an EOC Workstation.

Desired Response:

The RMS software will determine that its host is an EOC Workstation and instantiate the objects that comprise the Workstation RMS object model in order to function in the necessary role.

Pre-Conditions:

An operational database will need to be established, complete and available for the Workstation RMS process.

Post-Conditions:

Users will be able to view ground telemetry pages that display RMS provided information about available logical strings.

Users will be able to issue service requests to establish logical strings for monitoring a Real-Time contact, simulation of a real-time contact, or replay of historical telemetry.

Users will be able to connect to established real-time strings for monitoring of the real-time contact in mirrored or tailored telemetry processing mode.

3.2.4.1.3 Scenario Description

The RMS Controller's Initialization operation is invoked. The FdEvEventLogger object is created to allow RMS to send events to DMS. The CSMS IF object is created and the Controller determines, via the CSMS nameserver, whether it is on the RTS or on the WS. Upon determining that its host is a workstation, it creates the FdDsFileAccessor object and retrieves an Operational DB. The FrGrRmsWsRmsIF object will be created and will allow the WS RMS to receive objects from the RTS RMS. It will then create the Parameter Server process and register it with the nameserver. The FoPsClientIF object is created to allow the RMS to communicate with the Parameter Server. A RWSet object that will contain the IF objects for each RTS RMS will be created. In this particular scenario there are two RT Servers. Therefore, two FrGrWsRmsRequestProxy objects are created and added to the RWSet. The String Table is constructed and a FrGrTableUpdateRequest object is constructed to query each RTS RMS for its String Table objects. The FrGrTableUpdateRequest object is constructed and sent to the RTS RMS via the appropriate FrGrWsRmsRequestProxy object. The RTS RMS will send the Request back to the WS RMS with its String Table included. The String Table objects are retrieved from the Request and added to the WS RMS String Table. The String Table Update Request Status is received from the RTS RMS and the process is repeated in order to retrieve the second RTS RMS's String Table. Once the workstation string table is constructed, the Parameter Server is updated with the new string table parameters. The FrGrRequestHandler object is constructed in order to receive Requests from FUI, Analysis, and DMS.

3.2.4.1.4 State Transition Description

3.2.4.2 Initialization of RMS Residing on the Real-Time Server Scenario

3.2.4.2.1 Initialization of RMS Residing on the Real-Time Server Abstract

The purpose of the Initialization of RMS Residing on the Real-Time Server scenario is to describe the process by which the RMS is initialized upon system startup of a Real-Time Server.

3.2.4.2.2 Initialization of RMS Residing on the Real-Time Server Summary Information

Interfaces:

CSMS Name Server

Data Management Subsystem

Parameter Server Subsystem

Telemetry Subsystem

Command Subsystem

Real-Time Contact Management Subsystem

FUI Ground Script Controller

Stimulus:

The RMS software is executed on a Real-Time Server.

Desired Response:

The RMS software will determine that its host is a Real-Time Server and instantiate the objects that make up the RTS RMS object model in order to function in the necessary role.

Pre-Conditions:

An operational database will need to be established, complete and available for the RTS RMS process.

Post-Conditions:

Users will be able to issue service requests to establish logical strings for monitoring a real-time contact, simulation of a real-time contact, or replay of historical telemetry.

Users will be able to connect to established real-time strings for monitoring of the real-time contact in mirrored or tailored telemetry processing mode.

3.2.4.2.3 Scenario Description

The RMS Controller's Initialization operation is invoked. The FdEvEventLogger object is created in order for RMS to send events to DMS. The CSMS IF object is created and the Controller determines whether it is on the RTS or on the WS. Upon determining that its host is a RTS, it creates the FdDsFileAccessor object and retrieves an Operational DB. The FrGrRmsWsRmsIF

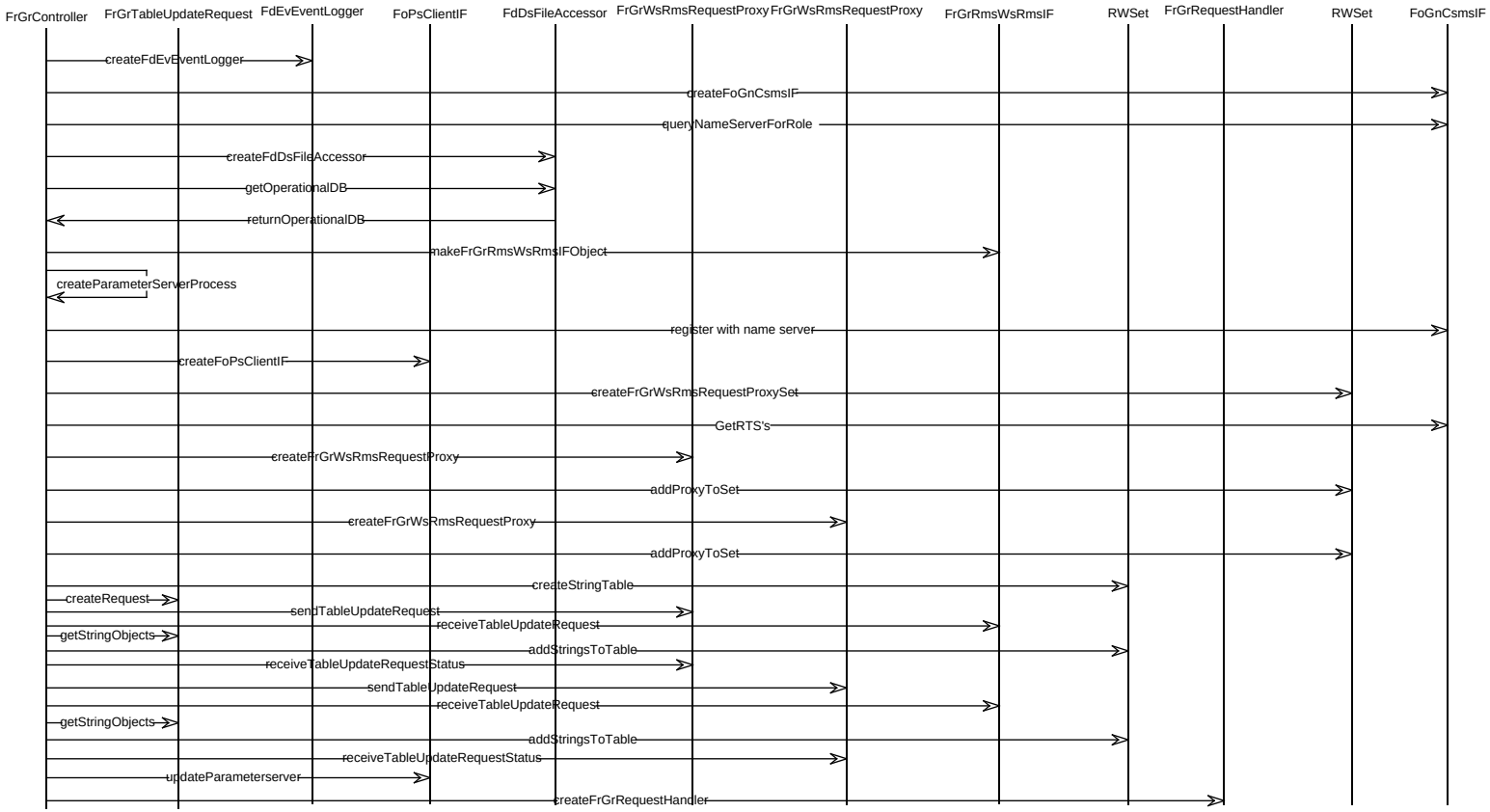


Figure 3.2.4.1.4-1. Initialization of RMS Residing on the Workstation Event Trace

object is created in order to receive Requests from the RMS processes that reside on the workstation. A RWSet collection class is created that will contain the Default Configuration Request objects. A default configuration file is requested from the DMS and loaded into the newly created default configuration collection class. A String Table is created as well as the FrGrStrManResMonProxy object. This object allows the RMS String Manager process to communicate with the RMS Resource Monitor process. An iterator is created that will allow the RMS to iterate over the Default Configuration Request objects. In this particular scenario the only Default Configuration Request is a single FrGrRealtimeServiceRequest. This request is executed. In doing so, a string object is made and added to the String Table. A FrGrTableUpdateRequest object is created and multicasted to each WS RMS in order for their String Table to be updated with the new String.

A FrGrParameterServer object is created in order for a Parameter Server process to be initialized. In Figure 3.2.4.2.4-3, the FrGrParameterServer object can be found. The Parameter Server process is created via the FrGrParameterServer object. In Figure 3.2.4.2.4-1 the ParameterServer object is added to the String and a Request is sent to the Resource Monitor to monitor the Parameter Server process.

A FrGrTelemetry object is created that is representative of the telemetry processes responsible for decommutation of the housekeeping, health&safety, standby, and diagnostic telemetry data. In Figure 3.2.4.2.4-3, four corresponding telemetry process objects are created. These telemetry process objects are representative of the processes used to decommutate housekeeping, health&safety, standby and diagnostic telemetry. The FrGrDataArchiver objects are created when the FrGrTelemetryProcess objects are created. An additional telemetry process object is created that is representative of the state check Telemetry Process. A FrGrDataArchiver object will not be created when this telemetry process object is created. The Request object notifies the FrGrTelemetry object to configure. FrGrTelemetry will notify its corresponding FrGrTelemetryProcess objects to start. If the FrGrTelemetryProcess represents a decommutation or dump process, the FrGrDataArchiver object is notified to start first. The data archiver process is created and the FrGrTelemetryProcess object creates the decommutation or dump process. Diagnostic telemetry will be decommutated via a dump telemetry process. Housekeeping, Health&Safety, and Standby telemetry will be decommutated via a decommutation telemetry process. If the FrGrTelemetryProcess represents the state check process, the data archiver process is not created and only the state check process is created. After the data archiver, decommutation, dump, and state check processes are created, the FrGrTelemetry object notifies each FrGrTelemetryProcess object to Config. If the FrGrTelemetryProcess object is associated with a FrGrDataArchiver object, the FrGrDataArchiver object will be notified by the FrGrTelemetryProcess object to Config. Once the data archiver process is configured, the decommutation and dump processes will be configured. In the case of configuring a state check process, only the state check process is configured. There is no data archiver process associated with the state check telemetry process. After all decommutation, diagnostic, state check, and data archiver processes are configured, the FrGrRealtimeServiceRequest is notified that telemetry has been configured. In Figure 3.2.4.2.4-1, the FrGrTelemetry object is added to the string. The RMS Resource Monitor process is notified of the new Process IDs that it needs to monitor.

In Figure 3.2.4.2.4-2, a FrGrCommand object is created that is representative of the command processes responsible for commanding the spacecraft. In Figure 3.2.4.2.4-4, three

FrGrCommandProcess objects are created. These objects will be used to communicate with the Transmit, Fop, and Format proxies. The Request object notifies FrGrCommand to configure. FrGrCommand notifies each FrGrCommandProcess object to start. A Transmit Command process is created and told the RMS address on the command line. The process will notify RMS that it is alive. This is repeated for the Format and Fop Command processes. After all command processes are started, the command object notifies the three FrGrCommandProcess objects to config. The command processes are sent their configuration information. In Figure 3.2.4.2.4-2, the Request object adds the FrGrCommand object to the string. The RMS Resource Monitor is notified of the command Process IDs that it needs to monitor.

A FrGrRTContact object is created that is representative of the RCM processes responsible for processing Nascom and EDOS data. In Figure 3.2.4.2.4-5, the FrGrRcmProcess objects are created. The FrGrDataArchiver object is created when a FrGrRcmProcess object is created. These objects will be used to communicate with the NoutMgr, EoutMgr, and DataArchiver proxies. The Request object notifies FrGrRTContact to configure. FrGrRTContact notifies the first FrGrRcmProcess to start. This FrGrRcmProcess object is representative of the NoutMgr RCM process. The DataArchiver process is created and followed by creation of the NoutMgr process. The FrGrRTContact object will then notify the FrGrRcmProcess object to configure. The FrGrRcmProcess object will notify the FrGrDataArchiver object to configure. After the DataArchiver process is configured, the NoutMgr process is configured. This is repeated for the EoutMgr process. Once all DataArchiver and RCM processes are configured, the Request object is notified. In Figure 3.2.4.2.4-2, the RTContact object is added to the string. The newly created Process ID's are sent to the RMS Resource Monitor process in order for them to be monitored by CSMS. The EinMgr and NinMgr process IDs are sent to Resource Monitor, but they are started by their corresponding EoutMgr and NoutMgr processes.

The Request object will create the FrGrGroundScriptController object. In Figure 3.2.4.2.4-5, the FrGrGroundScriptProcess object is notified to create a Ground Script Controller process. Once the process is created, it is notified to configure. In Figure 3.2.4.2.4-2, the FrGrGroundScriptProcess is added to the string and the Ground Script Controller process PID is sent to the Resource Monitor process in order for it to be monitored.

The FrGrRealtimeServiceRequest has completed processing and the FrGrController deletes the Request. The Default Configuration Information is checked for additional Requests. No Requests are found and the RTS RMS has been initialized.

3.2.4.2.4 State Transition Description

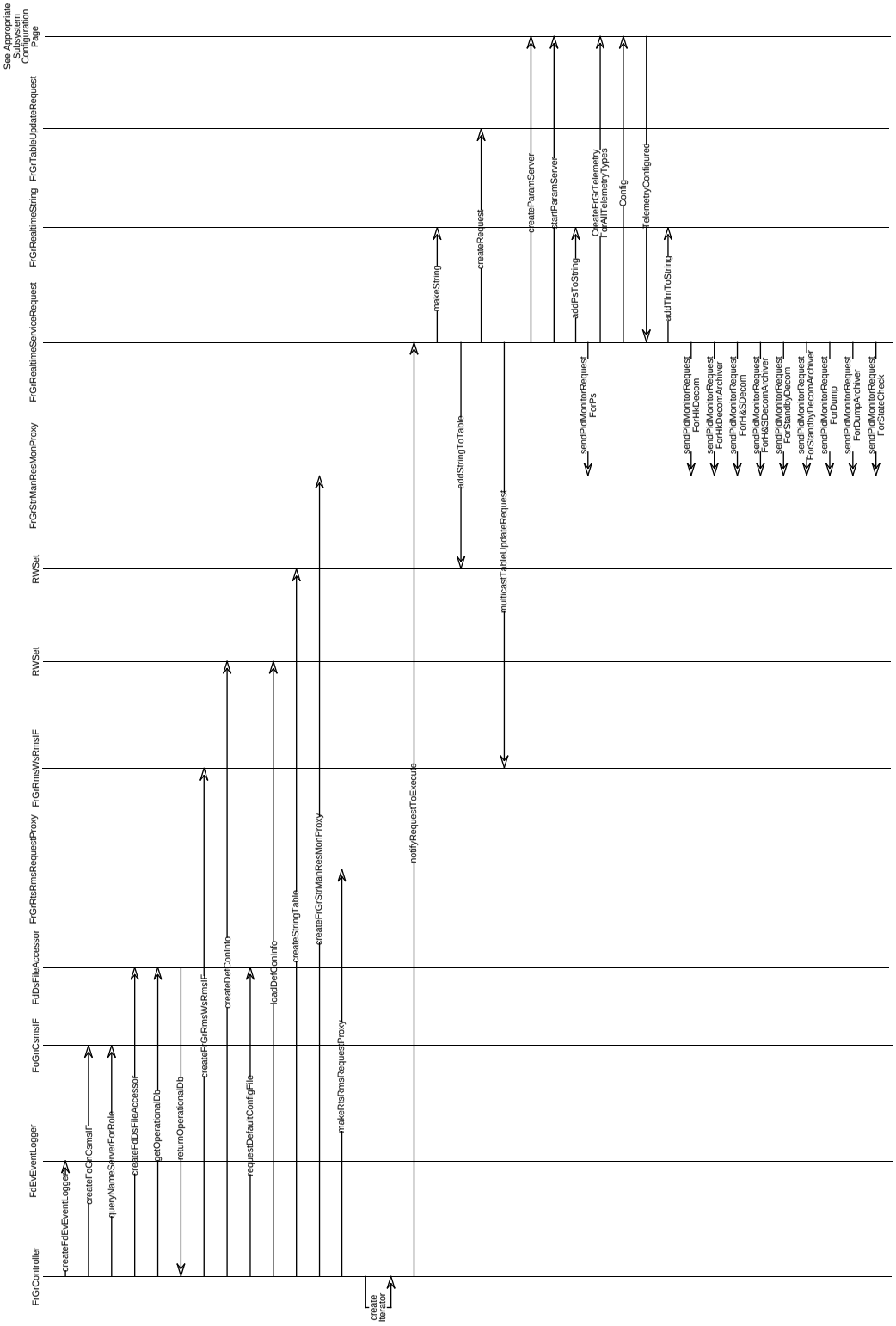


Figure 3.2.4.2.4-1. Initialization of RMS Residing on the Real-Time Server Event Trace (Part 1 of 2)

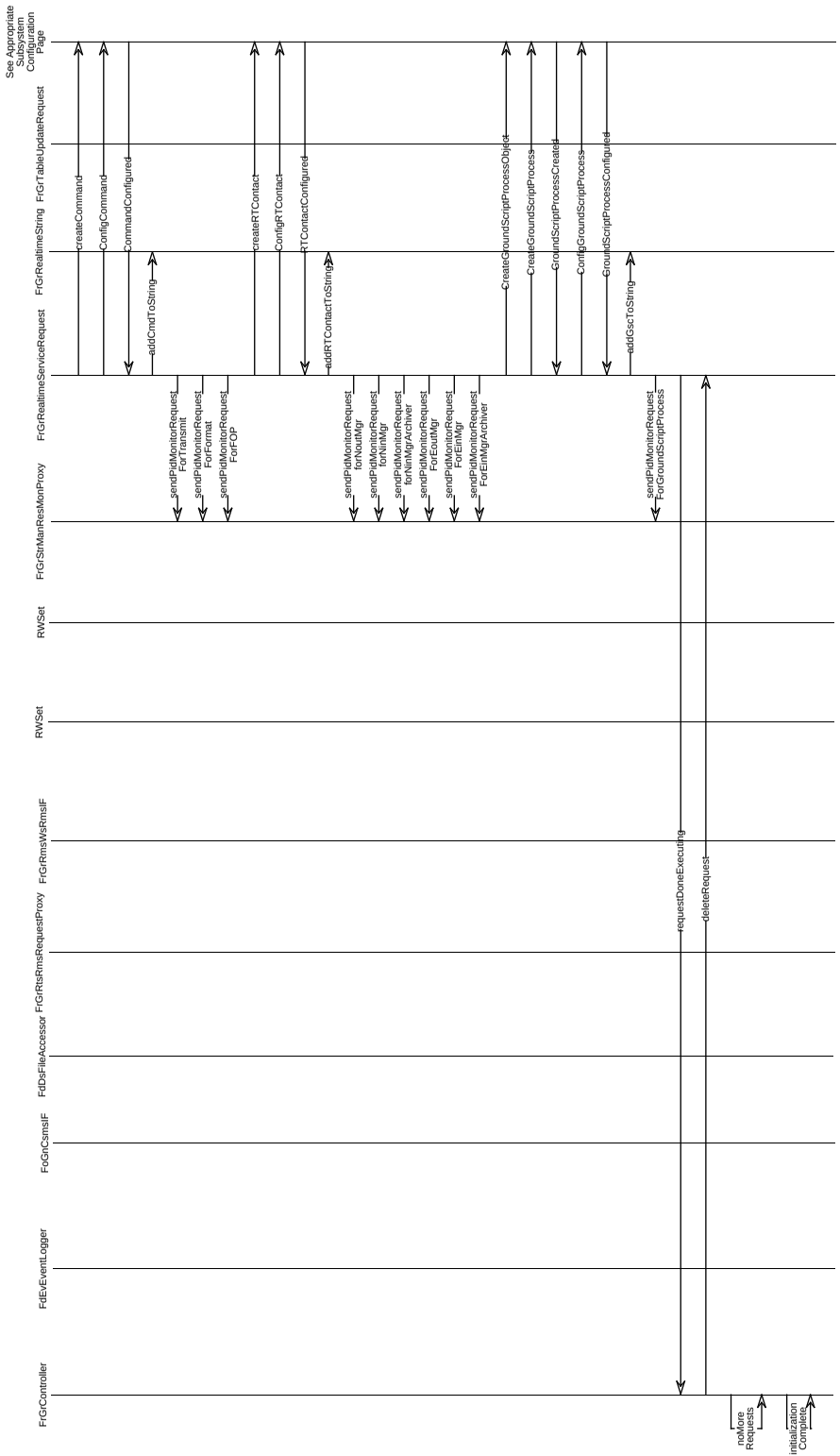


Figure 3.2.4.2.4-2. Initialization of RMS Residing on the Real-Time Server Event Trace (Part 2 of 2)

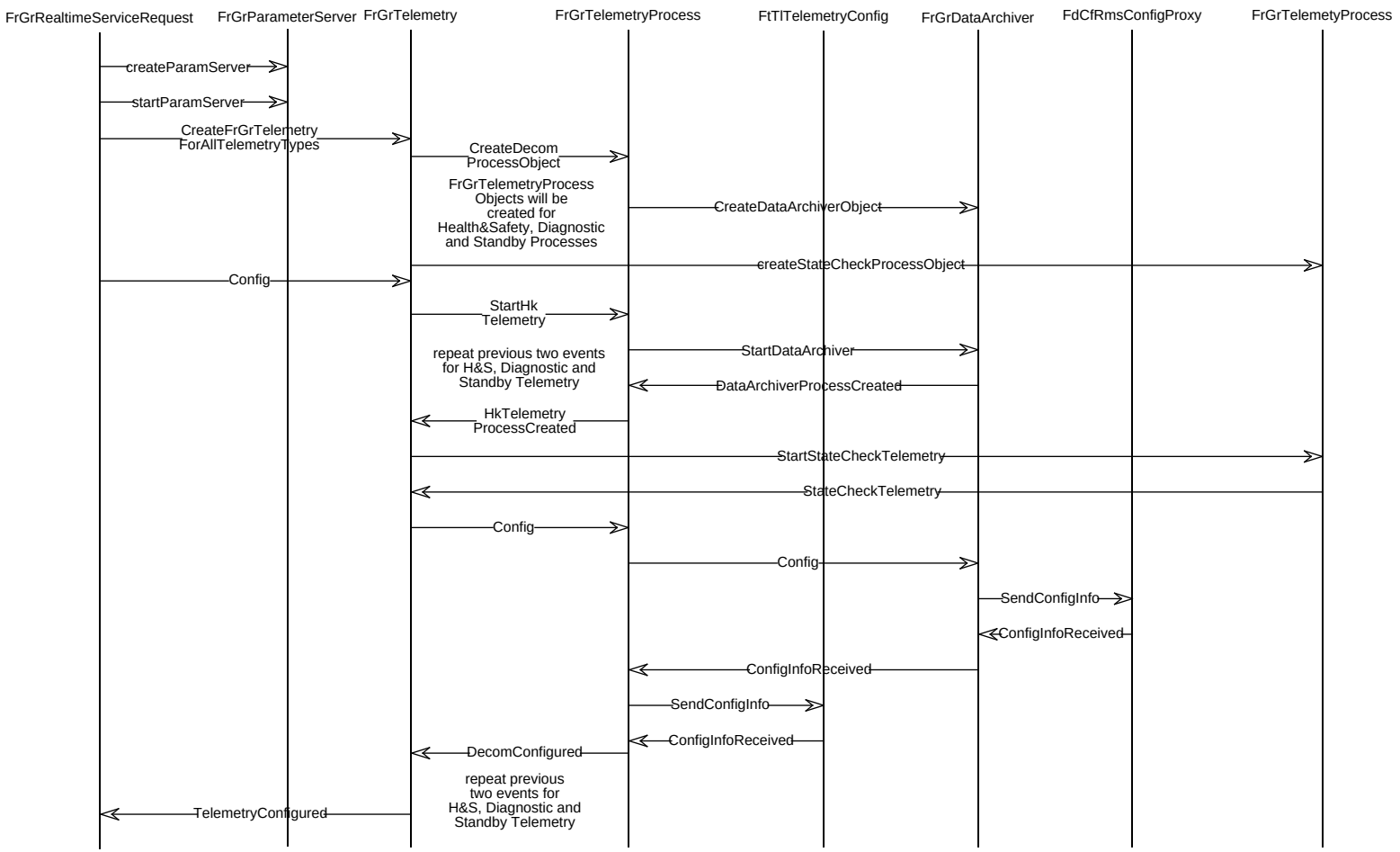


Figure 3.2.4.2.4-3. Initialization of RTS RMS - Parameter Server and Telemetry Subsystem Event Trace

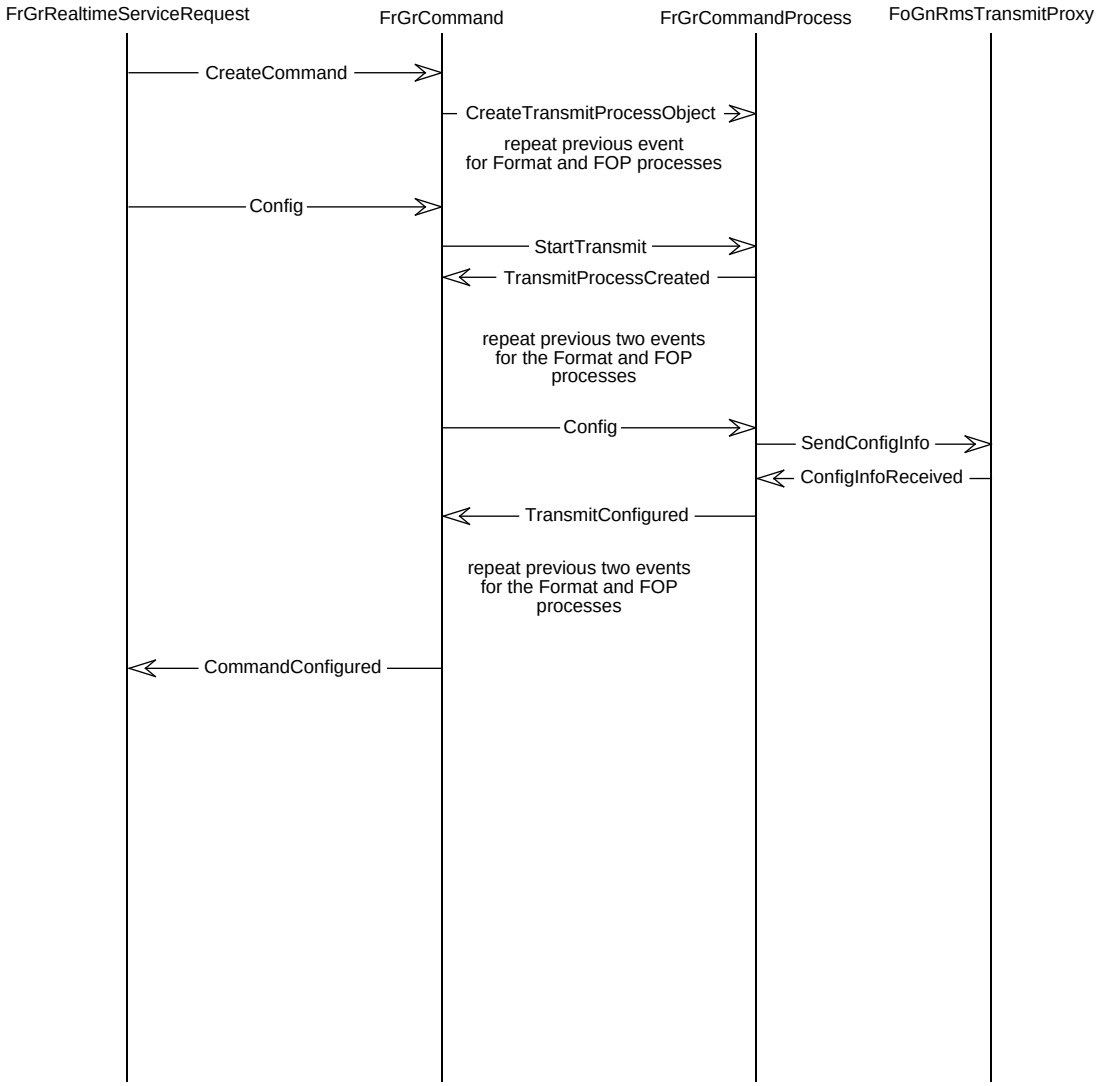


Figure 3.2.4.2.4-4. Initialization of RTS RMS - Command Subsystem Event Trace

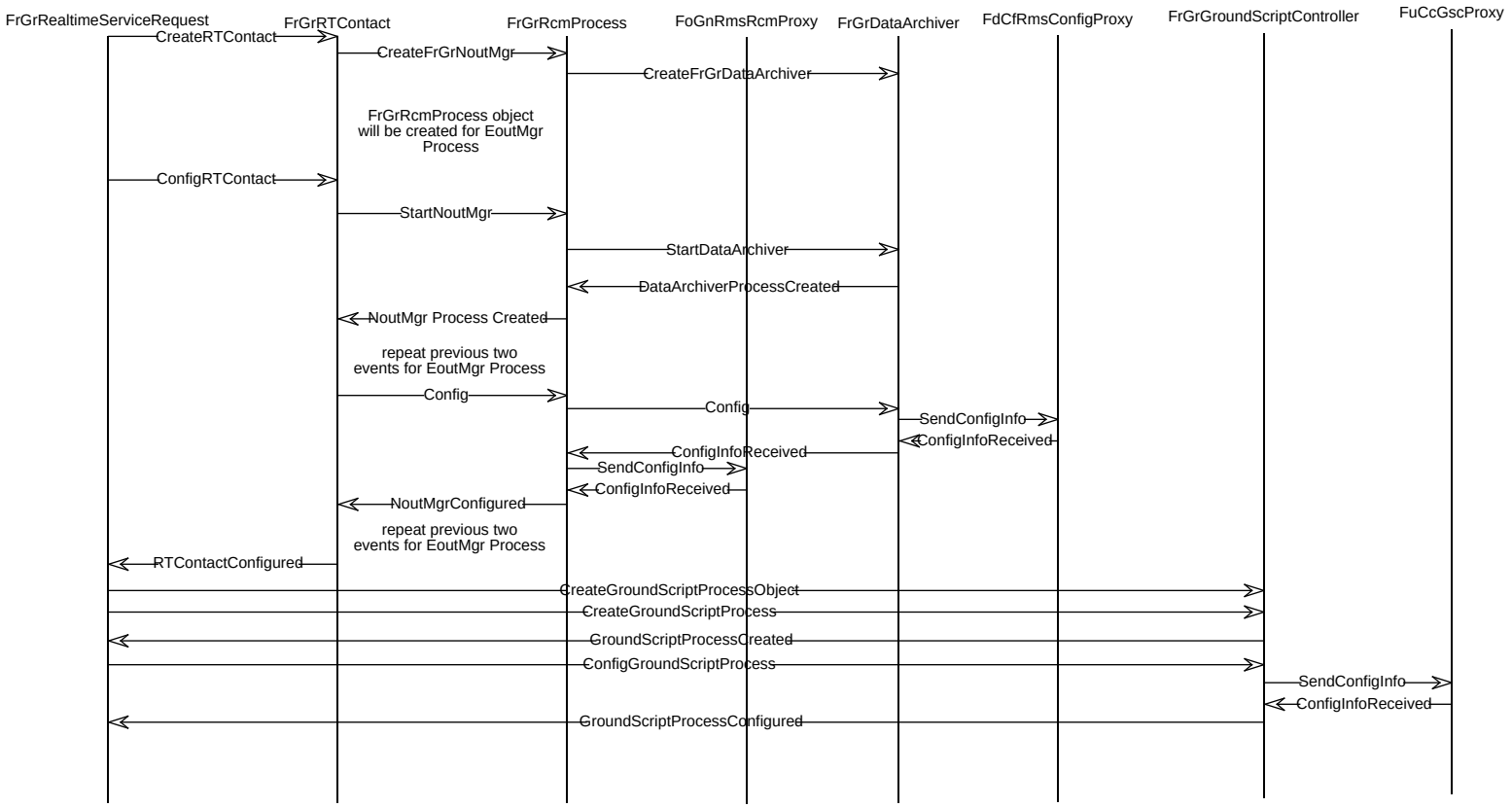


Figure 3.2.4.2.4-5. Initialization of RTS RMS - Real-Time Contact Management Subsystem and Ground Script Controller Event Trace

3.2.4.3 Request for A Real-Time Service Arrives on the Workstation Scenario

3.2.4.3.1 Request for a Real-Time Service Arrives on the Workstation Abstract

The purpose of the Request for a Real-Time Service Arrives on the Workstation scenario is to describe the process by which the Workstation RMS acts upon a request for a real-time service.

3.2.4.3.2 Request for a Real-Time Service Arrives on the Workstation Summary Information

Interfaces:

FOS User Interface Subsystem

SCDO

Stimulus:

The user wishing to create a Realtime String on the RTS, sends a RealtimeServiceRequest via the FUI.

Desired Response:

The WS RMS software will forward the Request to the RTS RMS to create a real-time string that includes all appropriate software associated with a real-time string.

Pre-Conditions:

An operational database has been established. The database will be used by string processes to retrieve configuration information.

Post-Conditions:

Users will be able to connect to the string for monitoring of the real-time contact in mirrored or tailored telemetry processing mode.

Users will be able to request Ground Control Authority in order to change the configuration of the string software processes located on the RTS and send Ground Control Message Requests (GCMRs) via the RCM software.

Users will be able to request Command Authority in order to send commands to the spacecraft via the string Ground Script Controller and Command processes.

EDOS CODA reports will be archived by DMS.

NASCOM blocks and EDOS Data Units (EDUs) will be archived by DMS.

3.2.4.3.3 Scenario Description

The Controller checks the queue of the FrGrRequestHandler and a FrGrRealtimeServiceRequest is returned. The Controller invokes the FrGrRealtimeServiceRequest object's Execute operation. It is then ensured that the request came from a valid Ground Controller and Ground Controller workstation. Upon determining that the string does not already exist and a DbId was supplied with the Request, the FrGrRealtimeServiceRequest is passed to the RTS RMS. When the request has been processed on the RTS a status is sent back to the FUI and the request is deleted.

3.2.4.3.4 State Transition Description

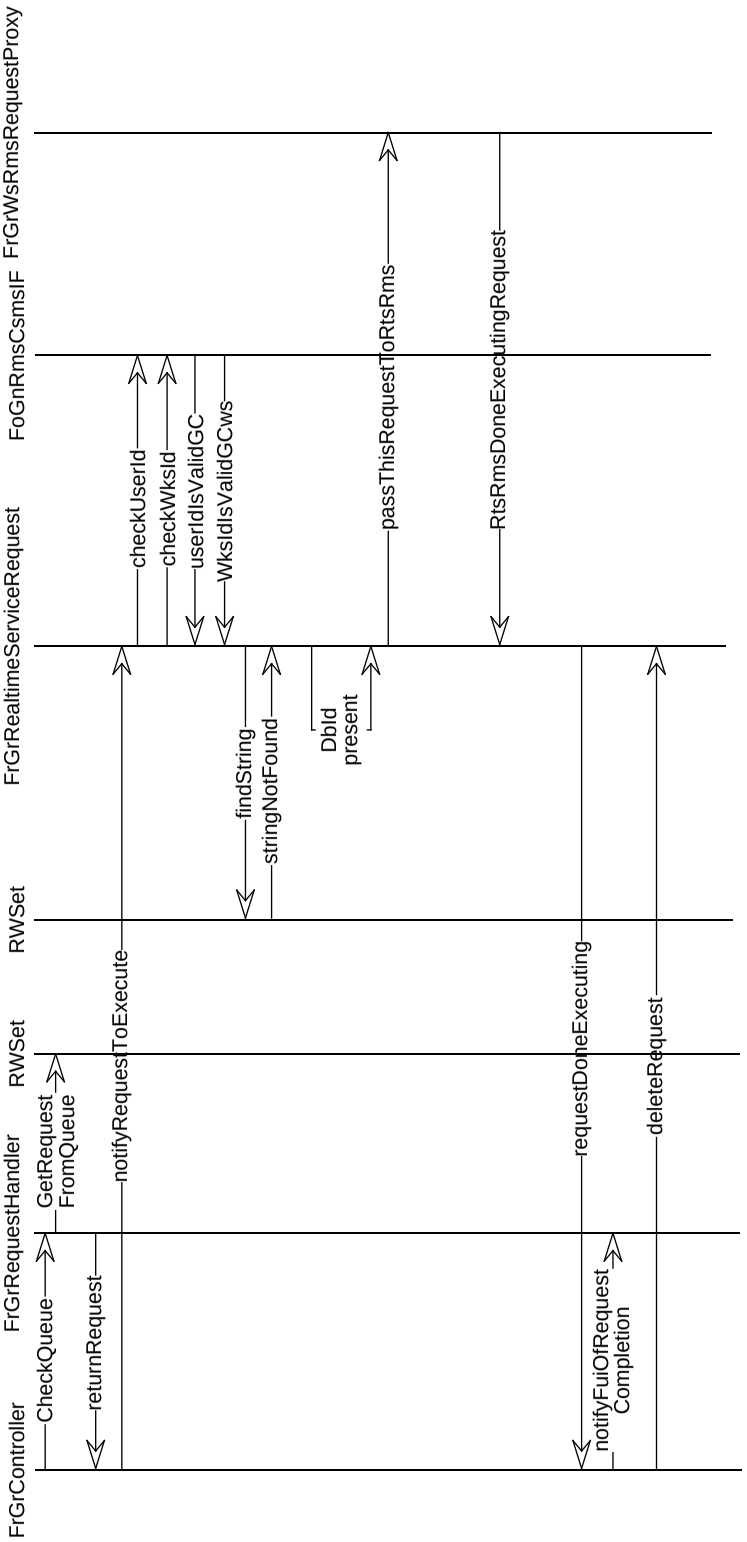


Figure 3.2.4.3.4-1. Request for a Real-Time Service Arrives on the Workstation Event Trace

3.2.4.4 Request for Real-Time Service Arrives on the Real-Time Server Scenario

3.2.4.4.1 Request for a Real-Time Service Arrives on the Real-Time Server Abstract

The purpose of the Request for Real-Time Service Arrives on the Real-Time Server scenario is to describe how the RTS RMS acts upon a user request for a real-time service that is forwarded from a Workstation RMS.

3.2.4.4.2 Request for a Real-Time Service Arrives on the Real-Time Server Summary Information

Interfaces:

Data Management Subsystem

Parameter Server Subsystem

Telemetry Subsystem

Command Subsystem

Real-Time Contact Management Subsystem

FUI Ground Script Controller

Stimulus:

The RMS on the workstation forwards a FrGrRealtimeServiceRequest object to the RMS on the RTS for processing.

Desired Response:

The RTS RMS software will create a real-time string that includes all appropriate software associated with a real-time string.

Pre-Conditions:

An operational database has been established. The database will be used by string processes to retrieve configuration information.

Post-Conditions:

Telemetry processes will be configured for decommutation of housekeeping, health&safety, standby, and dump telemetry data. The state check telemetry process will be started.

The Format, FOP, and Transmit Command processes will be configured.

The Ground Script Controller Process will be configured.

The RCM processes will be configured.

The Data Archiver processes will be configured.

The Parameter Server process will be created.

3.2.4.4.3 Scenario Description

The Controller checks the queue of the FrGrFrGrRmsWsRmsIF object and a FrGrRealtimeServiceRequest is returned. The Controller invokes the FrGrRealtimeServiceRequest object's Execute operation. A FrGrRealtimeString is created and

added to the String Table. A FrGrTableUpdateRequest object is created and multicasted to each WS RMS in order for their String Table to be updated with the new String.

A FrGrParameterServer object is created in order for a Parameter Server process to be initialized. In Figure 3.2.4.4.4-3, the FrGrParameterServer object can be found. The Parameter Server process is created via the FrGrParameterServer object. In Figure 3.2.4.4.4-1 the ParameterServer object is added to the String and a Request is sent to the Resource Monitor to monitor the Parameter Server process.

A FrGrTelemetry object is created that is representative of the telemetry processes responsible for decommutation of the housekeeping, health&safety, standby, and diagnostic telemetry data. In Figure 3.2.4.4.4-3, four corresponding telemetry process objects are created. These telemetry process objects are representative of the processes used to decommutate housekeeping, health&safety, standby and diagnostic telemetry. The FrGrDataArchiver objects are created when the FrGrTelemetryProcess objects are created. An additional telemetry process object is created that is representative of the state check Telemetry Process. A FrGrDataArchiver object will not be created when this telemetry process object is created. The Request object notifies the FrGrTelemetry object to configure. FrGrTelemetry will notify its corresponding FrGrTelemetryProcess objects to start. If the FrGrTelemetryProcess represents a decommutation or dump process, the FrGrDataArchiver object is notified to start first. The data archiver process is created and the FrGrTelemetryProcess object creates the decommutation or dump process. Diagnostic telemetry will be decommutated via a dump telemetry process. Housekeeping, Health&Safety, and Standby telemetry will be decommutated via a decommutation telemetry process. If the FrGrTelemetryProcess represents the state check process, the data archiver process is not created and only the state check process is created. After the data archiver, decommutation, dump, and state check processes are created, the FrGrTelemetry object notifies each FrGrTelemetryProcess object to Config. If the FrGrTelemetryProcess object is associated with a FrGrDataArchiver object, the FrGrDataArchiver object will be notified by the FrGrTelemetryProcess object to Config. Once the data archiver process is configured, the decommutation and dump processes will be configured. In the case of configuring a state check process, only the state check process is configured. There is no data archiver process associated with the state check telemetry process. After all decommutation, diagnostic, state check, and data archiver processes are configured, the FrGrRealtimeServiceRequest is notified that telemetry has been configured. In Figure 3.2.4.4.4-1, the FrGrTelemetry object is added to the string. The RMS Resource Monitor process is notified of the new Process IDs that it needs to monitor.

In Figure 3.2.4.4.4-2, a FrGrCommand object is created that is representative of the command processes responsible for commanding the spacecraft. In Figure 3.2.4.4.4-4, three FrGrCommandProcess objects are created. These objects will be used to communicate with the Transmit, Fop, and Format proxies. The Request object notifies FrGrCommand to configure. FrGrCommand notifies each FrGrCommandProcess object to start. A Transmit Command process is created and told the RMS address on the command line. The process will notify RMS that it is alive. This is repeated for the Format and Fop Command processes. After all command processes are started, the command object notifies the three FrGrCommandProcess objects to config. The command processes are sent their configuration information. In Figure 3.2.4.4.4-2, the Request object adds the FrGrCommand object to the string. The RMS Resource Monitor is notified of the command Process IDs that it needs to monitor.

A FrGrRTContact object is created that is representative of the RCM processes responsible for processing Nascom and EDOS data. In Figure 3.2.4.4.4-5, the FrGrRcmProcess objects are created. The FrGrDataArchiver object is created when a FrGrRcmProcess object is created. These objects will be used to communicate with the NoutMgr, EoutMgr, and DataArchiver proxies. The Request object notifies FrGrRTContact to configure. FrGrRTContact notifies the first FrGrRcmProcess to start. This FrGrRcmProcess object is representative of the NoutMgr RCM process. The DataArchiver process is created and followed by creation of the NoutMgr process. The FrGrRTContact object will then notify the FrGrRcmProcess object to configure. The FrGrRcmProcess object will notify the FrGrDataArchiver object to configure. After the DataArchiver process is configured, the NoutMgr process is configured. This is repeated for the EoutMgr process. Once all DataArchiver and RCM processes are configured, the Request object is notified. In Figure 3.2.4.4.4-2, the RTContact object is added to the string. The newly created Process ID's are sent to the RMS Resource Monitor process in order for them to be monitored by CSMS. The EinMgr and NinMgr process IDs are sent to Resource Monitor, but they are started by their corresponding EoutMgr and NoutMgr processes.

The Request object will create the FrGrGroundScriptController object. In Figure 3.2.4.4.4-5, the FrGrGroundScriptProcess object is notified to create a Ground Script Controller process. Once the process is created, it is notified to configure. In Figure 3.2.4.4.4-2, the FrGrGroundScriptProcess is added to the string and the Ground Script Controller process PID is sent to the Resource Monitor process in order for it to be monitored.

Once the FrGrRealtimeServiceRequest has completed processing, the FrGrController notifies the WS RMS of the Request Status and the Request is deleted.

3.2.4.4.4 State Transition Description

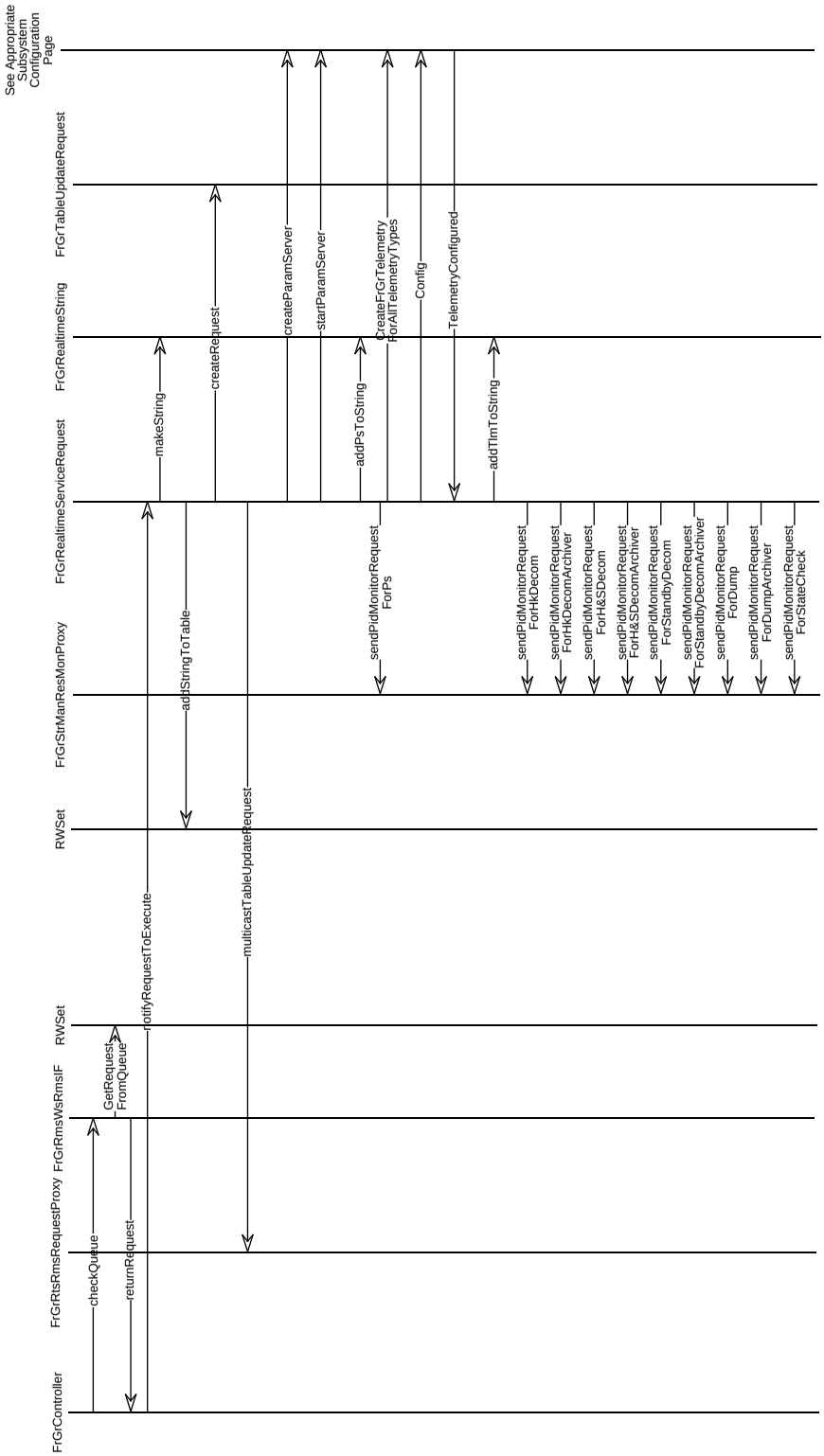


Figure 3.2.4.4.4-1. Request for a Real-Time Service Arrives on the Real-Time Server Event Trace (Part 1 of 2)

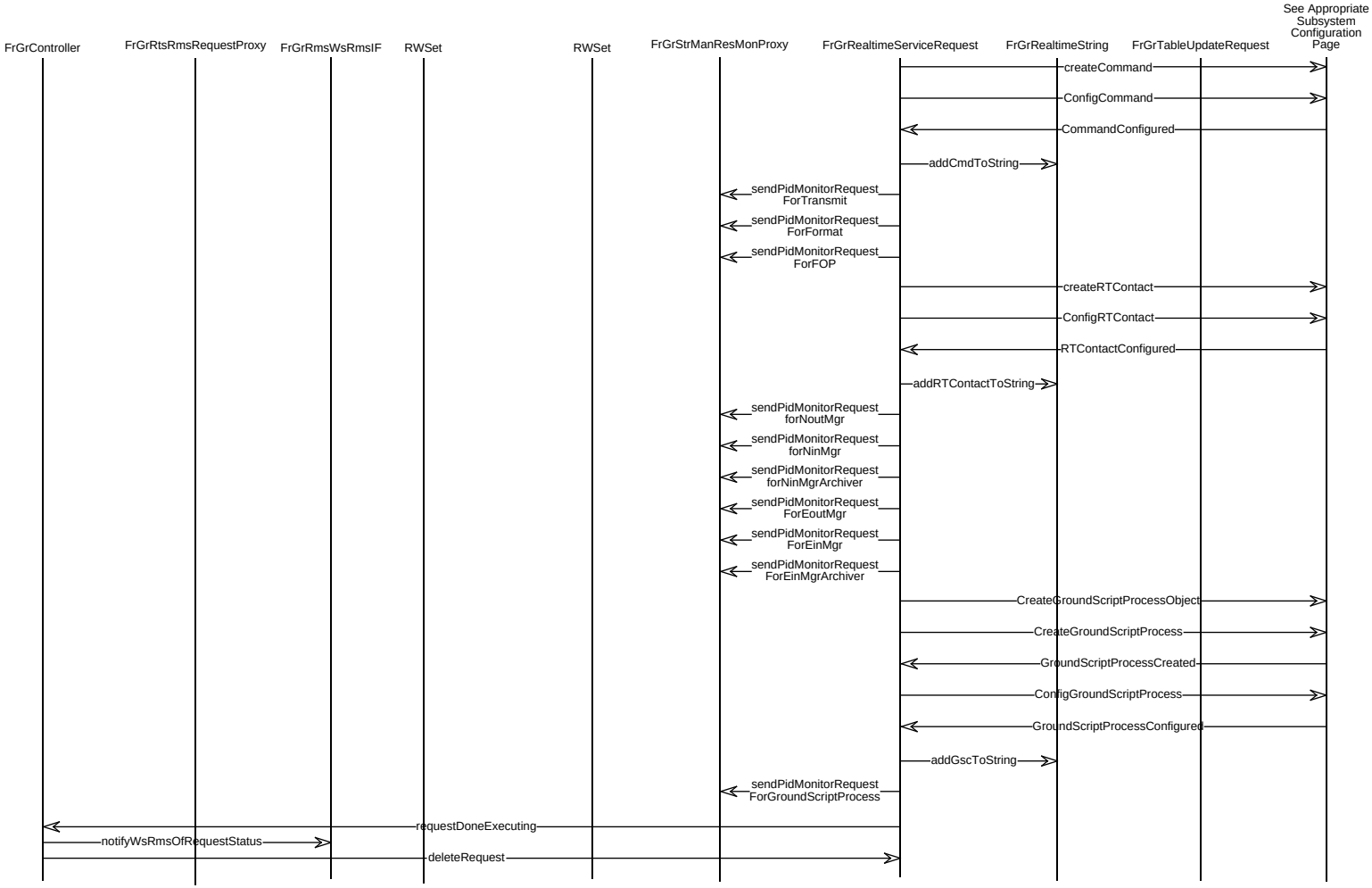


Figure 3.2.4.4.4-2. Request for a Real-Time Service Arrives on the Real-Time Server Event Trace (Part 2 of 2)

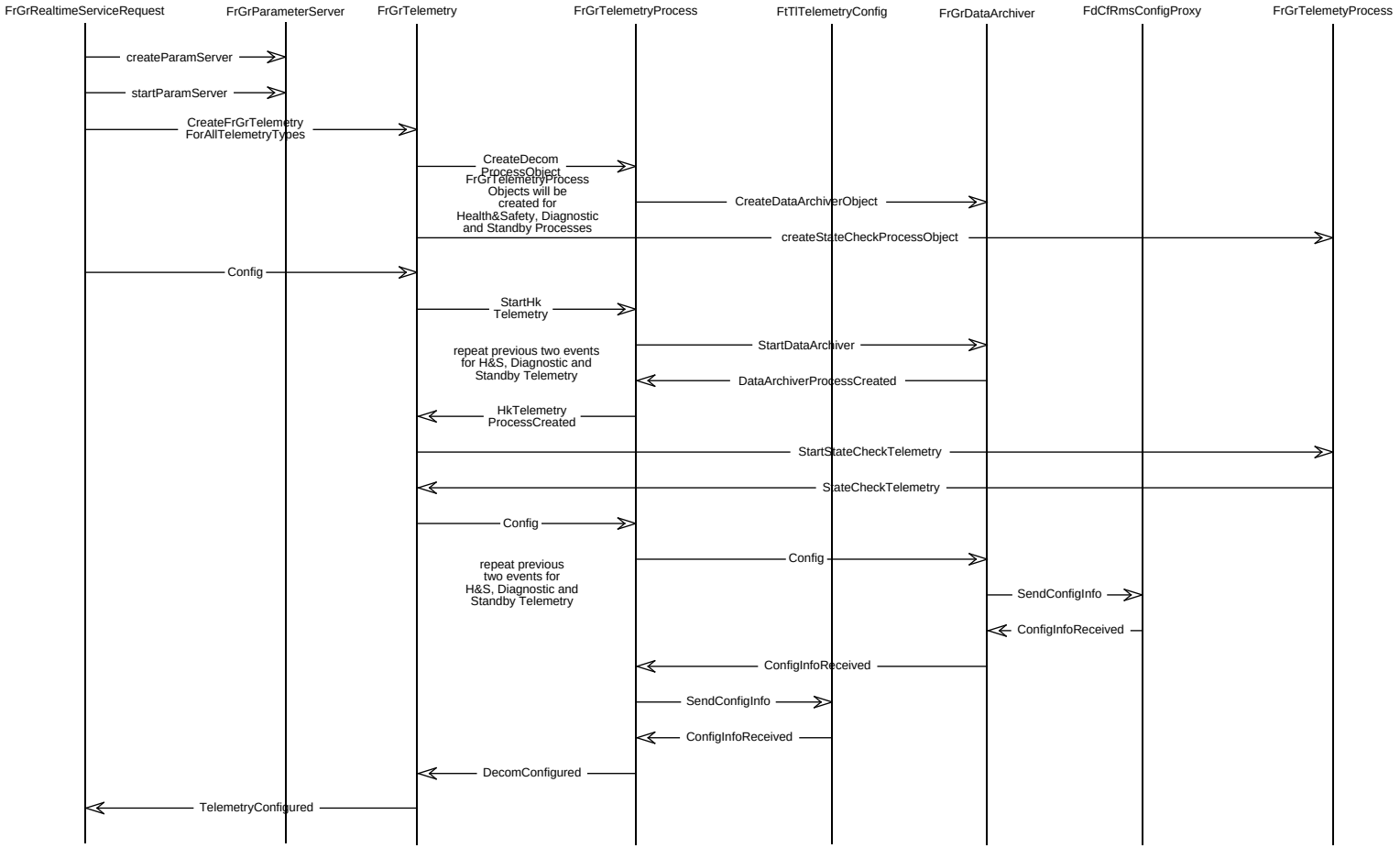


Figure 3.2.4.4.4-3. Request for a Real-Time Service - Parameter Server and Telemetry Subsystem Event Trace

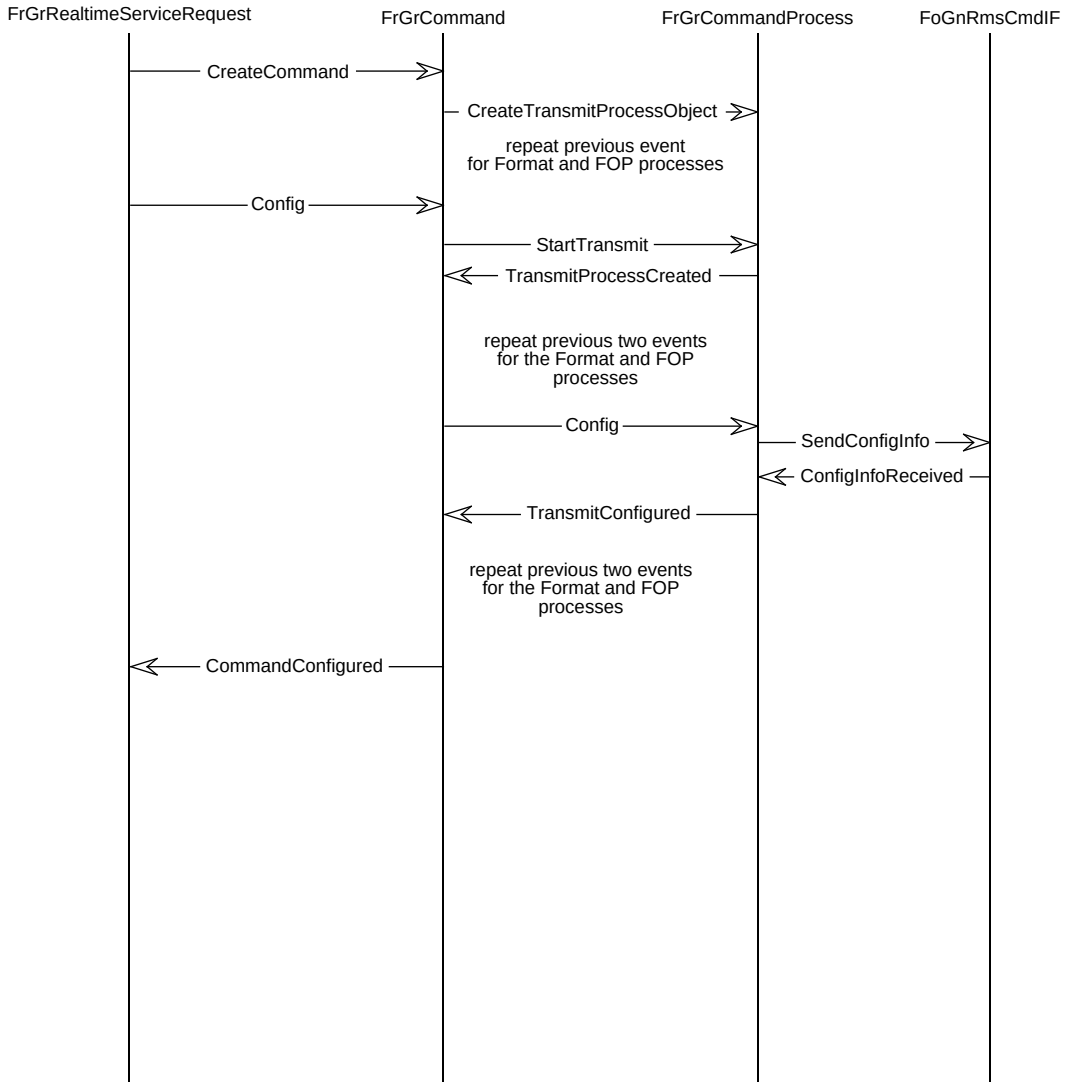


Figure 3.2.4.4.4-4. Request for a Real-Time Service - Command Subsystem Event Trace

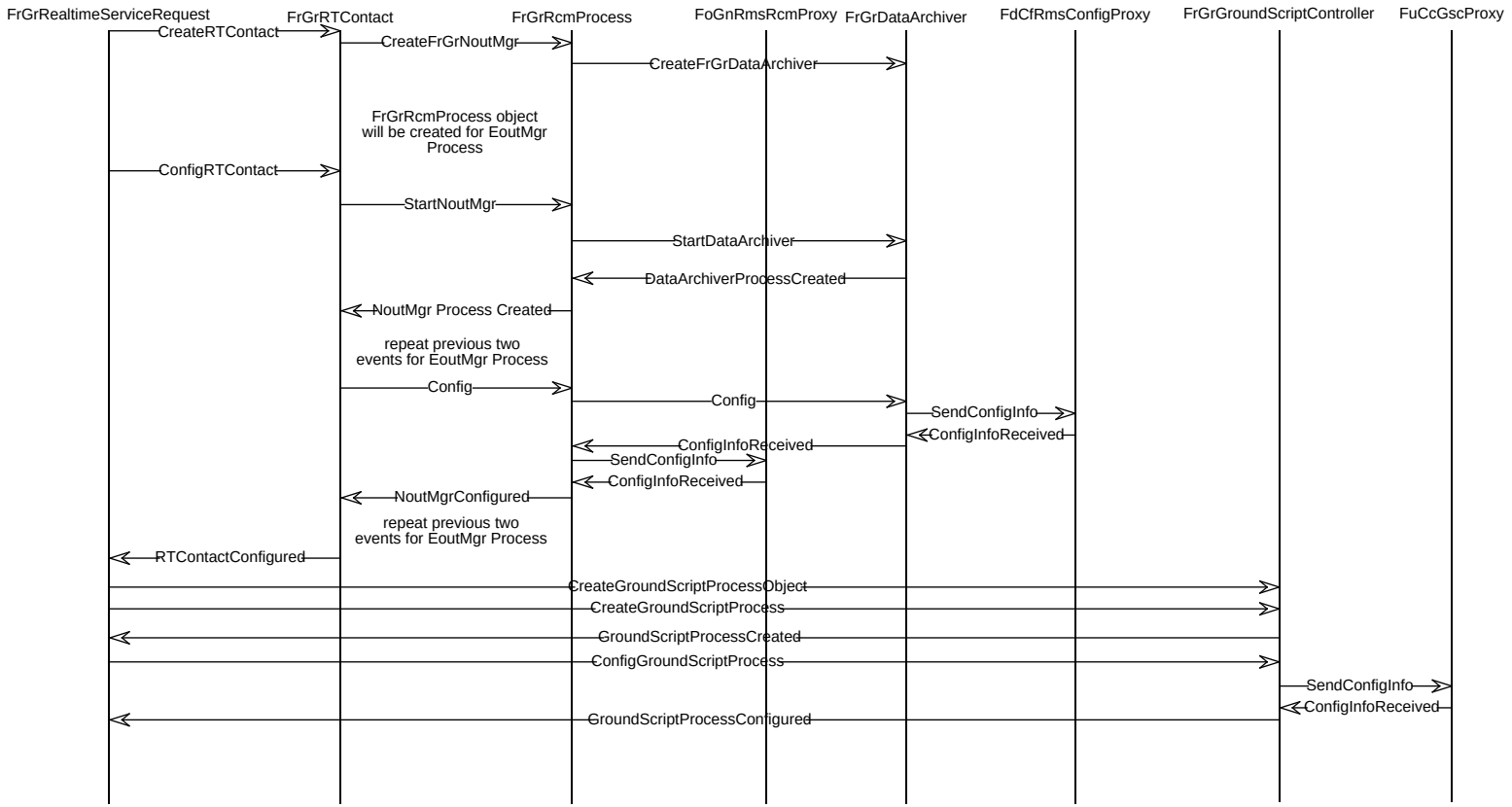


Figure 3.2.4.4.4-5. Request for a Real-Time Service - Real-Time Contact Management Subsystem and Ground Script Controller Event Trace

3.2.4.5 Request for a Mirrored String Connection Arrives on the Workstation Scenario

3.2.4.5.1 Request for a Mirrored String Connection Arrives on the Workstation Abstract

The purpose of the Request for a Mirrored String Connection Arrives on the Workstation scenario is to describe how the Workstation RMS acts upon a user request for a mirrored connection to an existing logical string.

3.2.4.5.2 Request for a Mirrored String Connection Arrives on the Workstation Summary Information

Interfaces:

FOS User Interface Subsystem

Parameter Server Subsystem

Telemetry Subsystem

Stimulus:

The user, wishing to connect to a logical string and mirror that string's telemetry process configuration on the RTS, sends a String Connect Request to the workstation RMS via the FUI.

Desired Response:

The RMS software will create a telemetry process on the WS with a configuration that mirrors the corresponding telemetry process on the RTS.

Pre-Conditions:

A string has been created for the user to connect to.

An operational database has been established. The database will be used by string processes to retrieve configuration information.

Post-Conditions:

The user will be able to monitor telemetry data using the same configuration as the RTS Telemetry Subsystem processes.

3.2.4.5.3 Scenario Description

The Controller checks the queue of the FrGrRequestHandler and a FrGrStringConnectRequest is returned. The Controller invokes the FrGrStringConnectRequest object's Execute operation. A string search is peformed and the string is found. It is determined that the user is not already connected. In Figure 3.2.4.5.4-2, the ParameterServer object is created. In Figure 3.2.4.5.4-1, the ParameterServer object is added to the string. In Figure 3.2.4.5.4-2, the Parameter Server process is started. This is followed by the creation of a FrGrTelemetry object and a FrGrTelemetryProcess object. In Figure 3.2.4.5.4-1, the newly created FrGrTelemetry object is added to the string. The FrGrStringConnectRequest is sent to the RTS RMS and the arrival of a Telemetry Configuration Snapshot is awaited. Once the FrGrSnapshotCompNotif object is received, via the

FrGrRmsWsRmsIF object, it is notified to execute. A string search is performed and the FrGrTelemetry object is retrieved from the string. The Notif object notifies the FrGrTelemetry object to configure. In Figure 3.2.4.5.4-2, the FrGrTelemetry object will notify its corresponding FrGrTelemetryProcess object to start. The FrGrTelemetryProcess object will create its process and tell it the RMS address on the command line. The process will notify RMS that it is alive. Once the process is created, the FrGrTelemetry object is notified. The FrGrTelemetry object will notify the FrGrTelemetryProcess object to configure with the Snapshot File. The FrGrTelemetryProcess object will send the snapshot file along with the other configuration information to the telemetry process. Once the telemetry process is configured, the FrGrSnapshotCompNotif object is notified. In Figure 3.2.4.5.4-1, the FrGrStringConnectRequest is notified that the FrGrSnapshotCompNotif object has completed execution and the FrGrStringConnectRequest deletes the FrGrSnapshotCompNotif object. The completion status of the original FrGrStringConnectRequest that was sent to the RTS RMS is received and the Controller is notified. FUI is notified of the String Connect Request completion and the FrGrStringConnectRequest object is deleted.

3.2.4.5.4 State Transition Description

3.2.4.6 Request for a Mirrored String Connection Arrives on the Real-Time Server Scenario

3.2.4.6.1 Request for a Mirrored String Connection Arrives on the Real-Time Server Abstract

The purpose of the Request for a Mirrored String Connection Arrives on the Real-Time Server scenario is to describe how the RTS RMS acts upon a request for a mirrored string connection that is forwarded from a workstation RMS.

3.2.4.6.2 Request for a Mirrored String Connection Arrives on the Real-Time Server Summary Information

Interfaces:

Telemetry Subsystem

Stimulus:

The RMS on the workstation forwards a FrGrStringConnectRequest object to the RTS RMS for processing.

Desired Response:

The RMS software will include the user as part of a string and request a configuration snapshot from a specific telemetry process. A Snapshot Completion Notification will be sent back to the WS RMS in order for the telemetry process on the WS to be configured in the same way as the telemetry on the RTS.

Pre-Conditions:

A string has been created for the user to connect to.

An operational database has been established. The database will be used by string processes to retrieve configuration information.

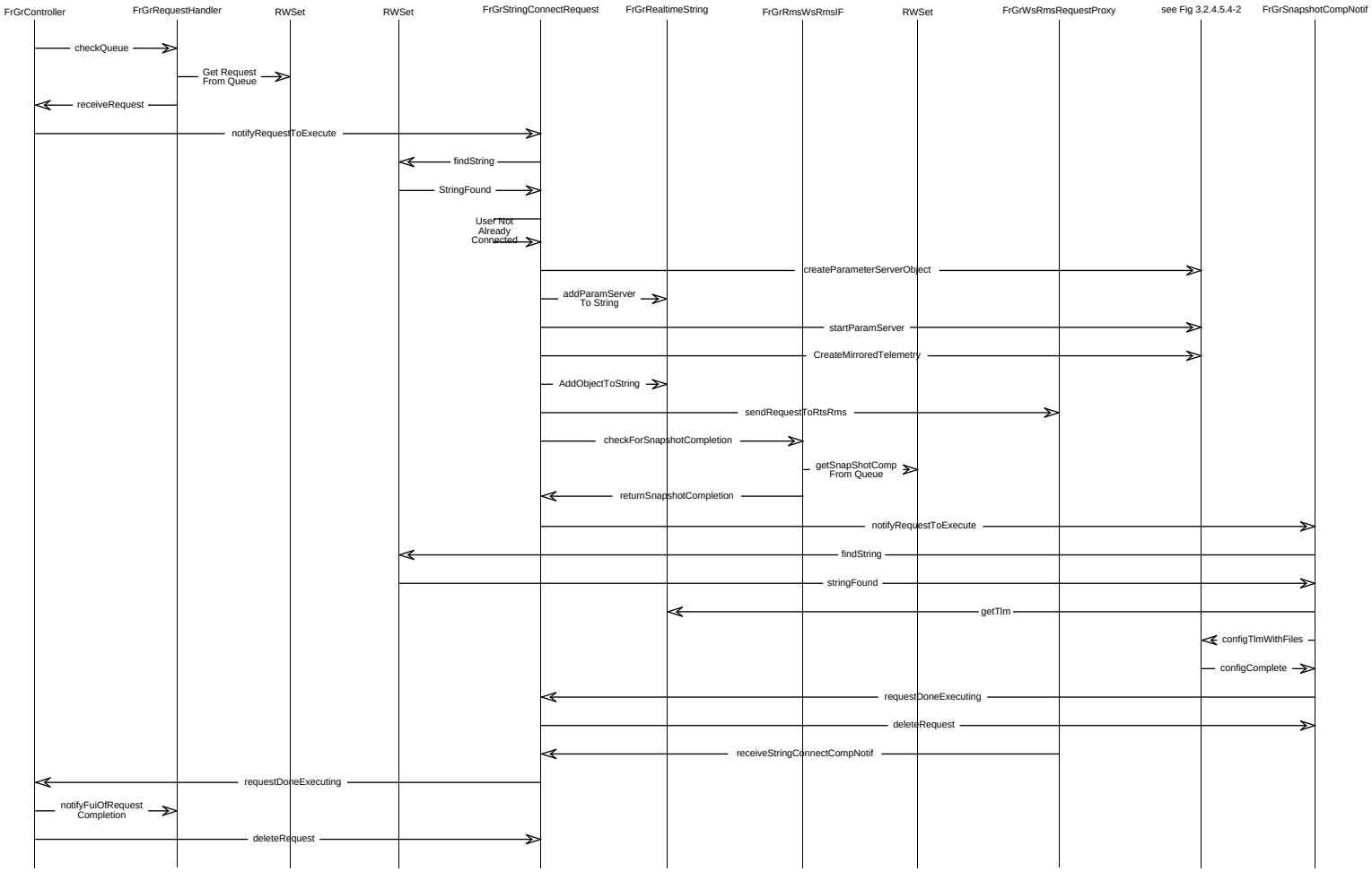


Figure 3.2.4.5.4-1. Execution of String Connection Request on the Workstation Event Trace

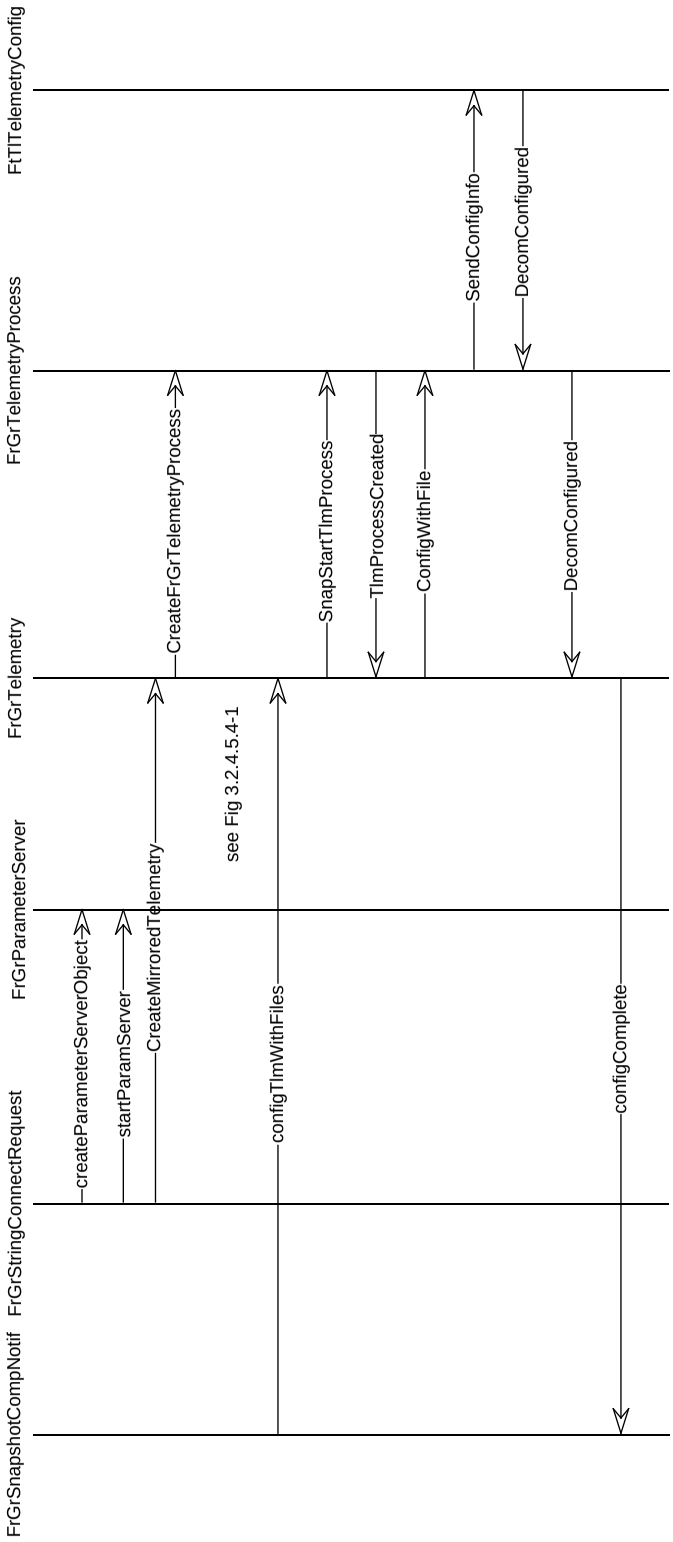


Figure 3.2.4.5.4-2. Creation of Mirrored Telemetry Subsystem on the Workstation Event Trace

Post-Conditions:

The configuration snapshot has been taken and the RTS RMS is capable of processing other requests from the WS RMS's.

3.2.4.6.3 Scenario Description

The Controller checks the queue of the FrGrRmsWsRmsIF and a FrGrStringConnectRequest is returned. The Controller invokes the FrGrStringConnectRequest object's Execute operation.

A string search is performed and the string is found. The user and workstation is added to the string and a FrGrTableUpdateRequest object is created. This object is multicasted to each WS RMS in order to update the WS RMS String Table. The appropriate FrGrTelemetry object is retrieved from the string and the FrGrTelemetry object is notified to Snap. In Figure 3.2.4.6.4-2, the corresponding FrGrTelemetryProcess object is then notified to send a Configuration Snapshot Request to Telemetry. The file is created by the Telemetry subsystem and is stored on the WS RMS under a predetermined path name. Once the Telemetry subsystem notifies the FrGrTelemetryProcess object that the snapshot has been taken, the FrGrTelemetry object is notified, followed by notification of the FrGrStringConnectRequest. In Figure 3.2.4.6.4-1, the FrGrSnapshotCompNotif object is created and sent to the WS RMS. The FrGrStringConnectRequest deletes the FrGrSnapshotCompNotif object and notifies the Controller that it has finished execution. After the Controller notifies the WS RMS that the StringConnectRequest has been processed, it deletes the FrGrStringConnectRequest.

3.2.4.6.4 State Transition Description

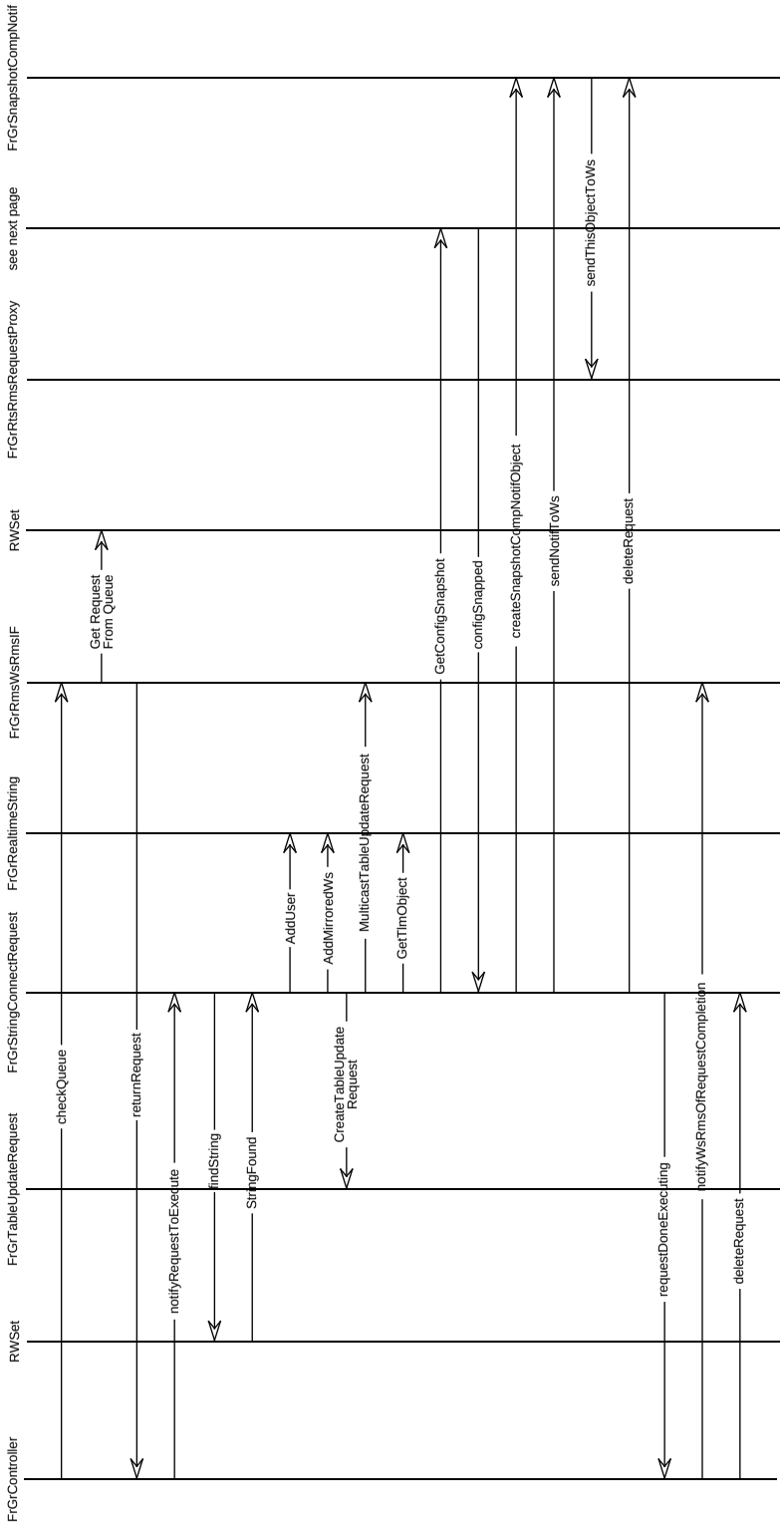


Figure 3.2.4.6.4-1. Execution of String Connection Request on the Real-Time Server Event Trace

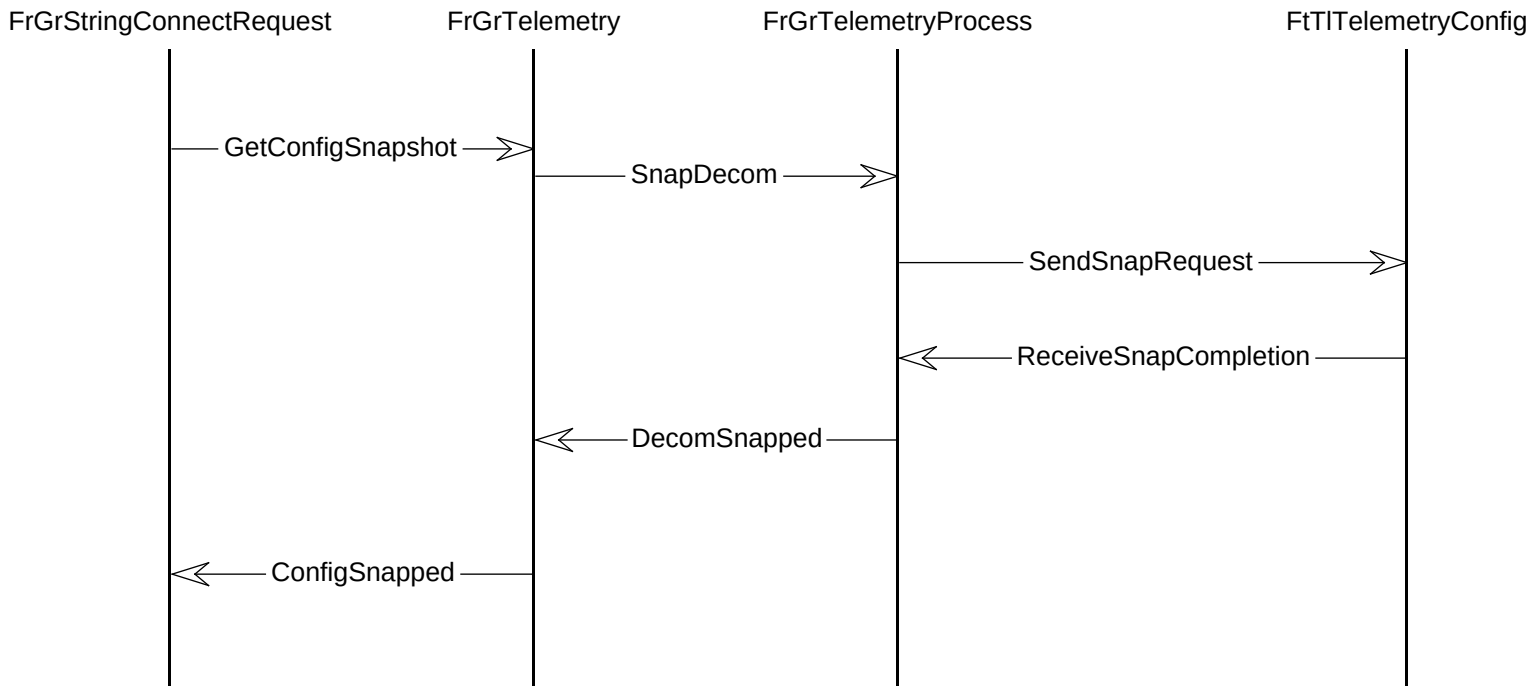


Figure 3.2.4.6.4-2. Creation of Telemetry Subsystem Configuration Snapshot on the Real-Time Server Event Trace

3.2.7.4 Request for Command Authority Arrives on the Workstation Scenario

3.2.4.7.1 Request for Command Authority Arrives on the Workstation Abstract

The purpose of the Request for Command Authority Arrives on the Workstation scenario is to describe how the workstation RMS acts upon a user request for command authority.

3.2.4.7.2 Request for Command Authority Arrives on the Workstation Summary Information

Interfaces:

SCDO/CSS Authorization Service

FOS User Interface

Stimulus:

The user, wishing to send real-time commands to a spacecraft associated with a particular string, sends a Command Privilege Request, via the user interface.

Desired Response:

The RMS software will determine if the user is eligible to receive the command authority privilege, and , if so, will forward the Request to the RTS RMS where the Command software will be notified of the new user with Command Authority.

Pre-Conditions:

A Real-Time or Simulation String will be available for the user to request Command Authority on.

Post-Conditions:

The user with Command Authority will be able to send commands to a spacecraft on a particular string.

3.2.4.7.3 Scenario Description

The Controller checks the queue of the FrGrRequestHandler and a FrGrCommandPrivilegeRequest is returned. The Controller invokes the CommandPrivilegeRequest object's Execute operation. A string search is performed and the string is found. Once it is determined that the string found is not a backup string, the CSMS is requested to validate the User and Workstation ID's as being valid for command authority. Validation is complete and the FrGrCommandPrivilegeRequest object is forwarded to the RTS RMS. The backup string is found and the same Request object is forwarded to the RTS RMS that is responsible for the backup string. Once both RTS RMS's have finished processing the CommandPrivilegeRequest, the FUI is notified and the FrGrCommandPrivilegeRequest object is deleted.

3.2.4.7.4 State Transition Description

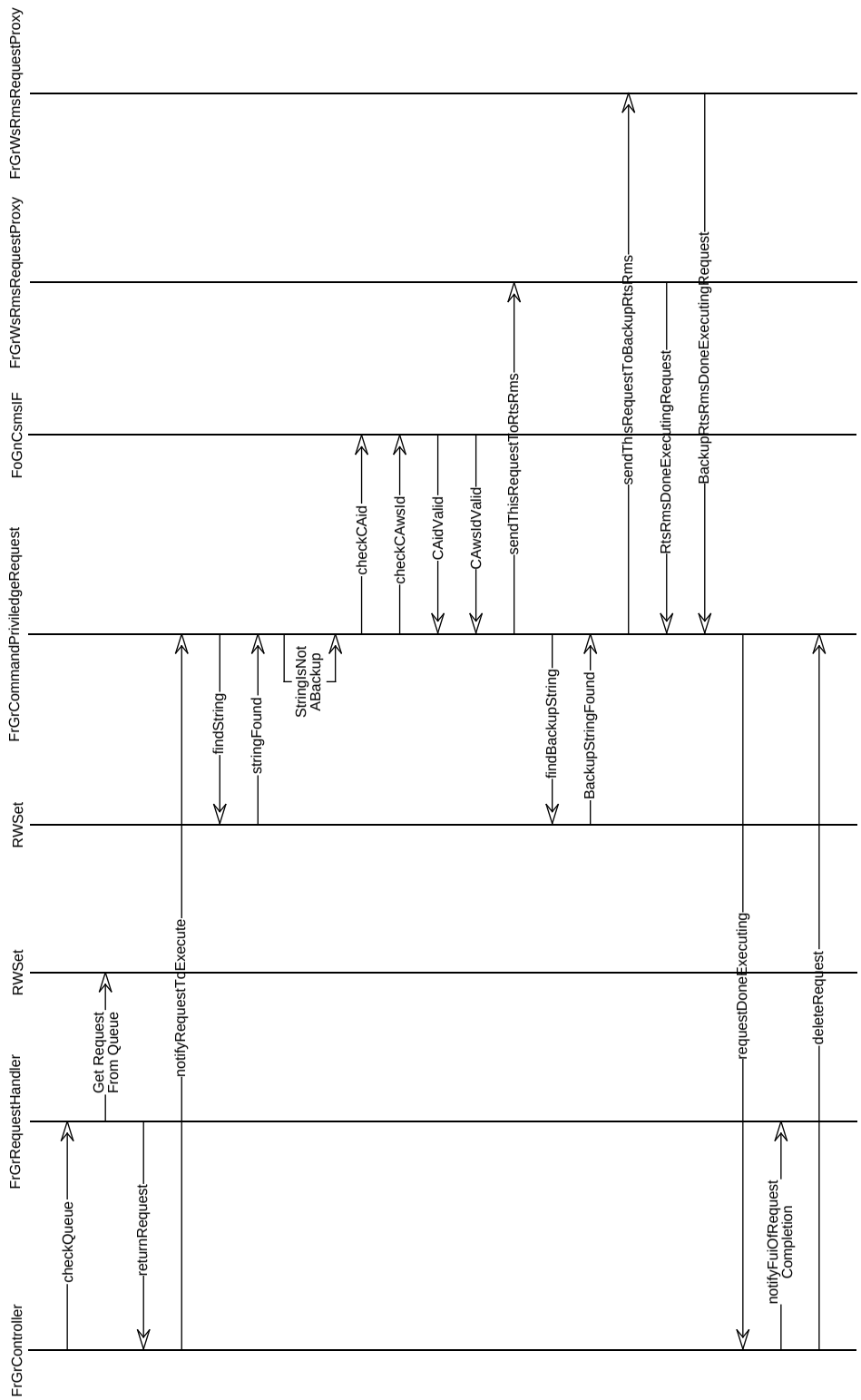


Figure 3.2.4.7.4-1. Request for Command Authority Arrives on the Workstation Event Trace

3.2.4.8 Request for Command Authority Arrives on the Real-Time Server Scenario

3.2.4.8.1 Request for Command Authority Arrives on the Real-Time Server Abstract

The purpose of the Request for Command Authority Arrives on the Real-Time Server scenario is to describe how a command authority change is sent to Command.

3.2.4.8.2 Request for Command Authority Arrives on the Real-Time Server Summary Information

Interfaces:

Command Subsystem

Stimulus:

The RMS on the workstation forwards a FrGrCommandPrivilegeRequest object to the RTS RMS for processing.

Desired Response:

The RMS software will notify the Command software of the new user with the Command Authority privilege.

Pre-Conditions:

A Real-Time or Simulation String will be available for the user to request Command Authority on.

Post-Conditions:

The Command software will be able to accept commands from a new user with the Command Authority privilege.

3.2.4.8.3 Scenario Description

The Controller checks the queue of the FrGrRmsWsRmsIF and a FrGrCommandPrivilegeRequest is returned The Controller invokes the FrGrCommandPrivilegeRequest object's Execute operation. A string search is performed and the string is found. The string's CAid and CAwsID is changed. The FrGrCommand object is retrieved from the string and the corresponding FrGrCommandProcess object is notified of the new CAid and CAwsID. The FrGrCommandProcess object sends this information to the Command process it is associated with. Once the Command process is reconfigured, the new Command Userstation ID is sent to the RMS Resource Monitor Task in order for it to begin monitoring the new CA userstation and stop monitoring the old CA userstation. A FrGrTableUpdateRequest is created and multicasted to each WS RMS in order for the WS RMS String Tables to be updated. The Controller is notified that the Request has finished processing and the WS RMS that sent the Request is notified. Finally, the Controller will delete the Request.

3.2.4.8.4 State Transition Description

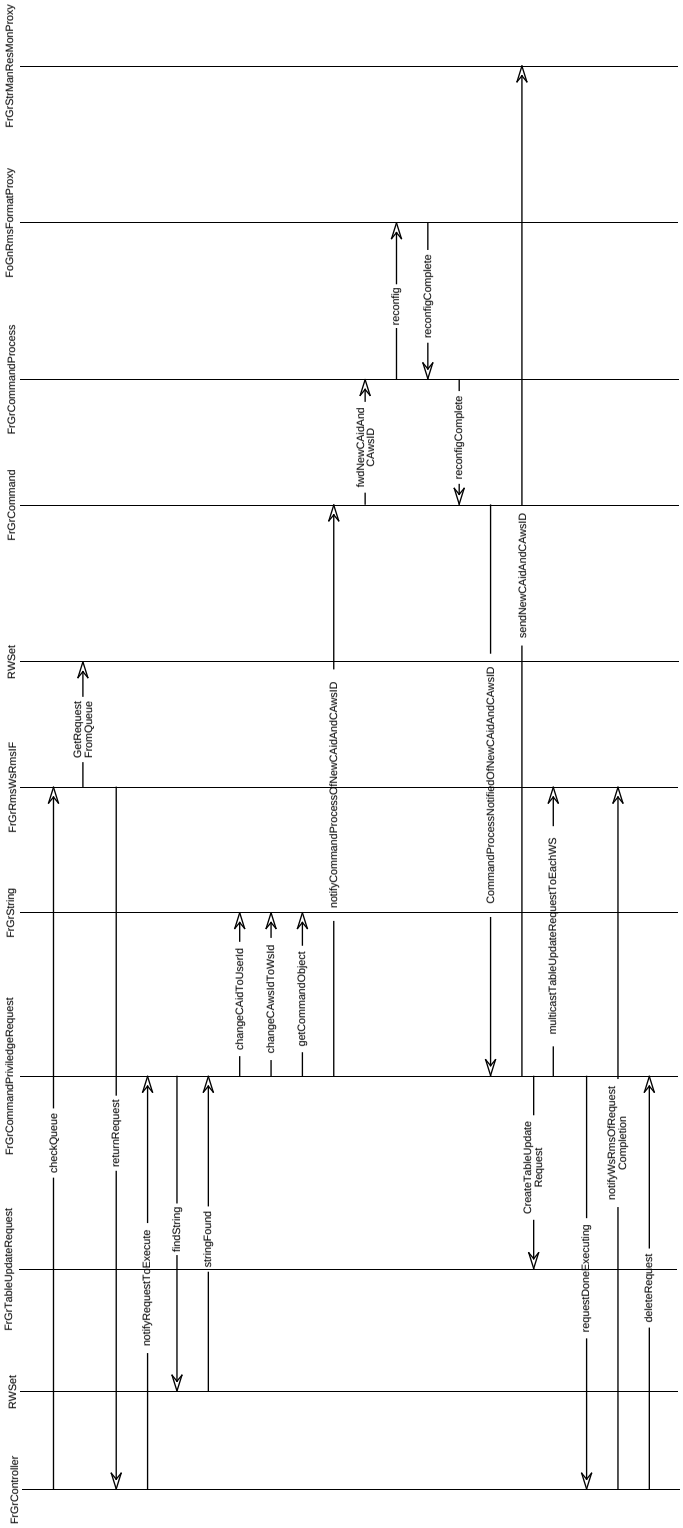


Figure 3.2.4.8.4-1. Request for Command Authority Arrives on the Real-Time Server Event Trace

3.2.4.9 Request for Telemetry Configuration Change Arrives on the Workstation Scenario

3.2.4.9.1 Request for Telemetry Configuration Change Arrives on the Workstation Abstract

The purpose of the Request for Telemetry Configuration Change Arrives on the Workstation scenario is to describe how the WS RMS acts upon a user request to change the configuration of the Telemetry Subsystem on the RTS.

3.2.4.9.2 Request for Telemetry Configuration Change Arrives on the Workstation Summary Information

Interfaces:

Telemetry Subsystem

FOS User Interface Subsystem

Stimulus:

The user, wishing to send a limit adjustment to a particular telemetry subsystem, sends an Adjust Limit Request, via the user interface.

Desired Response:

The RMS software will determine that the configuration change is for telemetry on the RTS and forward the Request to the RTS RMS if it came from a valid Ground Controller.

Pre-Conditions:

A string will have to already have been created in order to change its configuration.

A user will need to have the Ground Control Privilege before sending the Request. Otherwise, the Request will be rejected by the WS RMS.

Post-Conditions:

The telmetry processes on the RTS will be reconfigured to reflect the configuration change.

All mirrored telemetry processes on the userstation will be reconfigured to reflect the configuration change.

3.2.4.9.3 Scenario Description

The Controller checks the queue of the FrGrRequestHandler and a FrGrAdjustLimitRequest is returned. The Controller invokes the AdjustLimitRequest object's Execute operation. A string search is performed and a nondedicated string is found. The FrGrAdjustLimitRequest's mySendFlag attribute is checked to ensure that the Request arrived from the FUI and will need to be forwarded to the RTS RMS. The User ID and User WS ID is checked to ensure that the Request came from a valid Ground Controller. The Request is validated and the FrGrAdjustLimitRequest object is forwarded to the RTS RMS. A backup string is found and the FrGrAdjustLimitRequest object is forwarded to the RTS RMS responsible for the backup string. A second FrGrAdjustLimitRequest object is sent back from the RTS RMS in order for the telemetry on this particular workstation to be reconfigured to reflect the limit adjustment. The second

FrGrAdjustLimitRequest object is notified to execute by the original FrGrAdjustLimitRequest object. A string search is performed and the nondedicated string is found. The FrGrAdjustLimitRequest object's mySendFlag is checked and it ensures that this FrGrAdjustLimitRequest object will not be forwarded back to the RTS RMS. The FrGrTelemetry object is retrieved from the string and the corresponding telemetry processes are reconfigured via the FrGrTelemetryProcess object. The original FrGrAdjustLimitRequest object is notified that the second FrGrAdjustLimitRequest object has completed execution and the original FrGrAdjustLimitRequest object deletes the second FrGrAdjustLimitRequest object. The RTS RMS sends back the completion of the original FrGrAdjustLimitRequest and the FUI is notified that the AdjustLimitRequest has been processed. The FrGrAdjustLimitRequest is deleted.

3.2.4.9.4 State Transition Description

3.2.4.10 Request for Telemetry Configuration Change Arrives on the Real-Time Server Scenario

3.2.4.10.1 Request for Telemetry Configuration Change Arrives on the Real-Time Server Abstract

The purpose of the Request for Telemetry Configuration Change Arrives on the Real-Time server scenario is to describe how the telemetry configuration change is sent to the appropriate telemetry process.

3.2.4.10.2 Request for Telemetry Configuration Change Arrives on the Real-Time Server Summary Information

Interfaces:

Telemetry Subsystem

FOS User Interface Subsystem

Stimulus:

The RMS on the workstation forwards a FrGrAdjustLimitRequest object to the RTS RMS for processing.

Desired Response:

The RMS software will send the configuration change to the appropriate telemetry processes as well as all mirrored userstations.

Pre-Conditions:

A string will have to already have been created in order to change its configuration.

Post-Conditions:

The telmetry processes on the RTS will be reconfigured to reflect the configuration change.

All mirrored telemetry processes on the userstation will be reconfigured to reflect the configuration change.

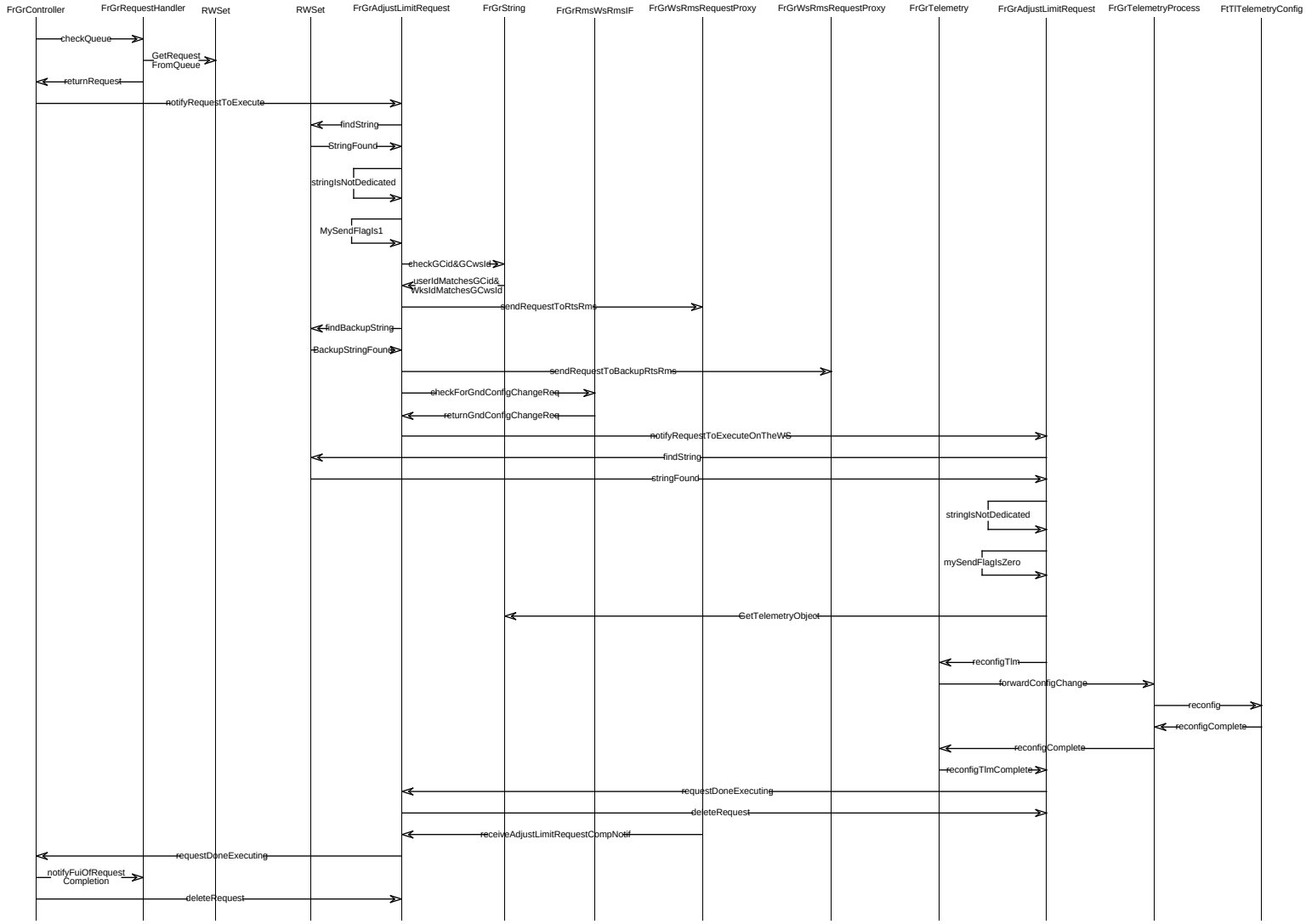


Figure 3.2.4.9.4-1. Request for Telemetry Configuration Change Arrives on the Workstation Event Trace

3.2.4.10.3 Scenario Description

The Controller checks the queue of the FrGrRmsWsRmsIF and a FrGrAdjustLimitRequest is returned. The Controller invokes the FrGrAdjustLimitRequest object's Execute operation. A string search is performed and the string is found. The FrGrTelemetry object is retrieved from the string and the corresponding telemetry processes are reconfigured, via the FrGrTelemetryProcess object, to reflect the limit adjustment. The mySendFlag attribute is set and the FrGrAdjustLimitRequest is sent back to each mirrored WS RMS in order for its telemetry process to be reconfigured. The WS RMS is notified that the AdjustLimitRequest has been processed and the FrGrAdjustLimitRequest object is deleted.

3.2.4.10.4 State Transition Description

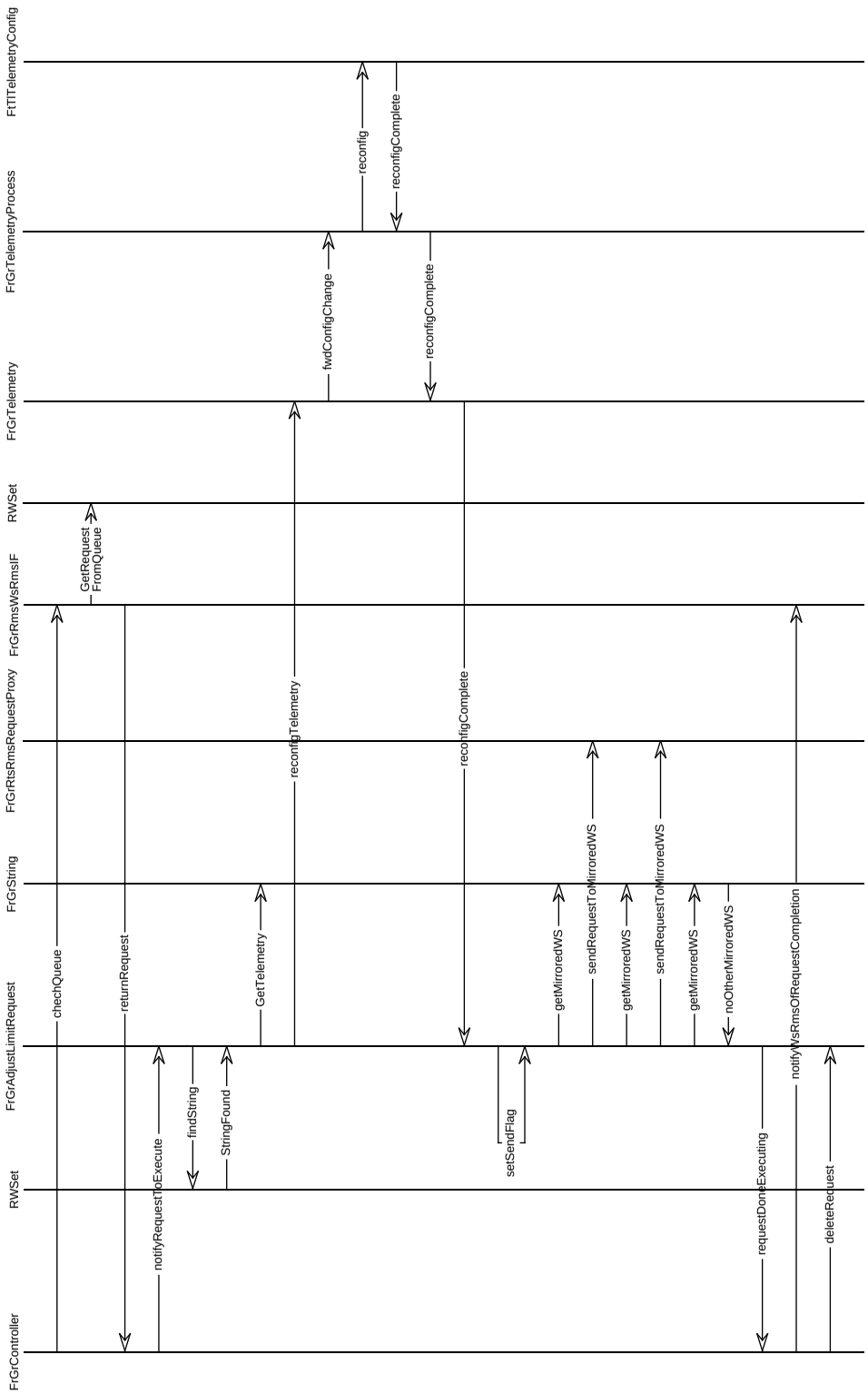


Figure 3.2.4.10.4-1. Request for Telemetry Configuration Change Arrives on the Real-Time Server Event

3.2.4.11 Request for Dedicated Replay Telemetry Arrives on the Workstation from DMS Scenario

3.2.4.11.1 Request for Dedicated Replay Telemetry Arrives on the Workstation from DMS Abstract

The purpose of the Request for Dedicated Replay Telemetry Arrives on the Workstation from DMS is to describe how the workstation RMS acts upon a user request for dedicated replay service.

3.2.4.11.2 Request for Dedicated Replay Telemetry Arrives on the Workstation from DMS Summary Information

Interfaces:

Parameter Server Subsystem

Telemetry Subsystem

Data Management Subsystem

Stimulus:

A user, wishing to establish a dedicated replay service on the workstation, enters a Replay Service Request on the command line. FUI, in turn, sends a Replay Service Request to the DMS. DMS determines if there are any database crossovers in the time period selected and sends this Request to the RMS.

Desired Response:

The RMS software will create a Dedicated Replay String on the userstation that includes a replay telemetry process.

Pre-Conditions:

The telemetry data to be replayed is made available by DMS.

Post-Conditions:

A telemetry process has been created and is capable of receiving replay data from DMS.

3.2.4.11.3 Scenario Description

The Controller checks the queue of the FrGrRequestHandler and a FrGrReplayServiceRequest is returned. The Controller invokes the ReplayServiceRequest object's Execute operation. The NULL value for the myRTSid attribute indicates that the Request is for dedicated replay telemetry. A string is made and added to the String Table. Since a DbId was provided with the Request, RMS will not have to query DMS for a DbId. A FrGrParameterServer object is created, added to the string, and the Parameter Server process is started. The FrGrTelemetry object is created and the corresponding FrGrTelemetryProcess object is created. The FrGrTelemetryProcess object will communicate with the telemetry process proxy. The FrGrTelemetry object is added to the string and the Request object will notify the FrGrTelemetry object to configure. FrGrTelemetry will notify the FrGrTelemetryProcess object to start. The FrGrTelemetryProcess object will create its process and tell it the RMS address on the command line. The process will notify RMS that it is alive. The FrGrTelemetry object will configure the telemetry process via the FrGrTelemetryProcess object. The Controller is notified that the Request has been processed and the DMS is notified of this as well. The FrGrReplayServiceRequest object is then deleted.

3.2.4.11.4 State Transition Description

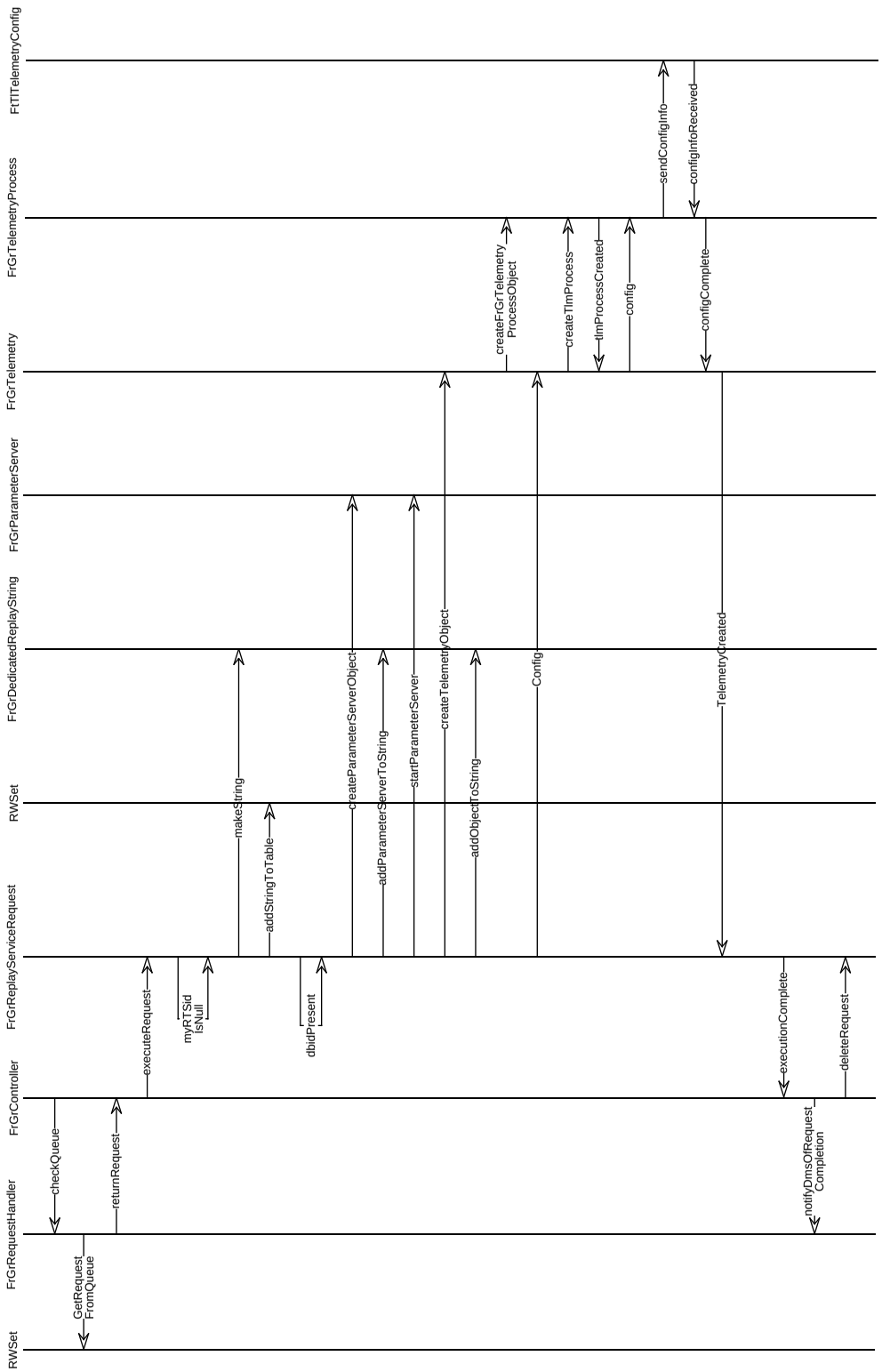


Figure 3.2.4.11.4-1 Request for Dedicated Replay Telemetry Arrives on the Workstation from DMS Event Trace

3.2.4.12 Request for Dedicated Replay Telemetry Arrives on the Workstation from Analysis Scenario

3.2.4.12.1 Request for Dedicated Replay Telemetry Arrives on the Workstation from Analysis Abstract

The purpose of the Request for Dedicated Replay Telemetry Arrives on the Workstation from Analysis scenario is to describe how the WS RMS acts upon a request for a dedicated replay service.

3.2.4.12.2 Request for Dedicated Replay Telemetry Arrives on the Workstation from Analysis Summary Information

Interfaces:

FOS Analysis Subsystem

Parameter Server Subsystem

Telemetry Subsystem

Stimulus:

Analysis, wishing to establish a dedicated replay telemetry process on the workstation, sends a Replay Service Request, via the Analysis Subsystem.

Desired Response:

The RMS software will create a Dedicated Replay String on the userstation that includes a replay telmetry process.

Pre-Conditions:

The telemetry data to be replayed is made available by DMS.

Post-Conditions:

A telemetry process has been created and is capable of receiving replay data from DMS.

3.2.4.12.3 Scenario Description

The Controller checks the queue of the FrGrRequestHandler and a FrGrReplayServiceRequest is returned. The Controller invokes the ReplayServiceRequest object's Execute operation. The NULL value for the myRTSid attribute indicates that the Request is for dedicated replay telemetry. A string is made and added to the String Table. Since a DbId was provided with the Request, RMS will not have to query DMS for a DbId. A FrGrParameterServer object is created, added to the string, and the Parameter Server process is started. The FrGrTelemetry object is created and the corresponding FrGrTelemetryProcess object is created. The FrGrTelemetryProcess object will communicate with the telemetry process proxy. The FrGrTelemetry object is added to the string and the Request object will notify the FrGrTelemetry object to configure. FrGrTelemetry will notify the FrGrTelemetryProcess object to start. The FrGrTelemetryProcess object will create its process and tell it the RMS address on the command line. The process will notify RMS that it is alive. The FrGrTelemetry object will configure the telemetry process via the FrGrTelemetryProcess object. The Controller is notified that the Request has been processed and the Analysis Subsystem is notified of this as well. The FrGrReplayServiceRequest object is then deleted.

3.2.4.12.4 State Transition Description

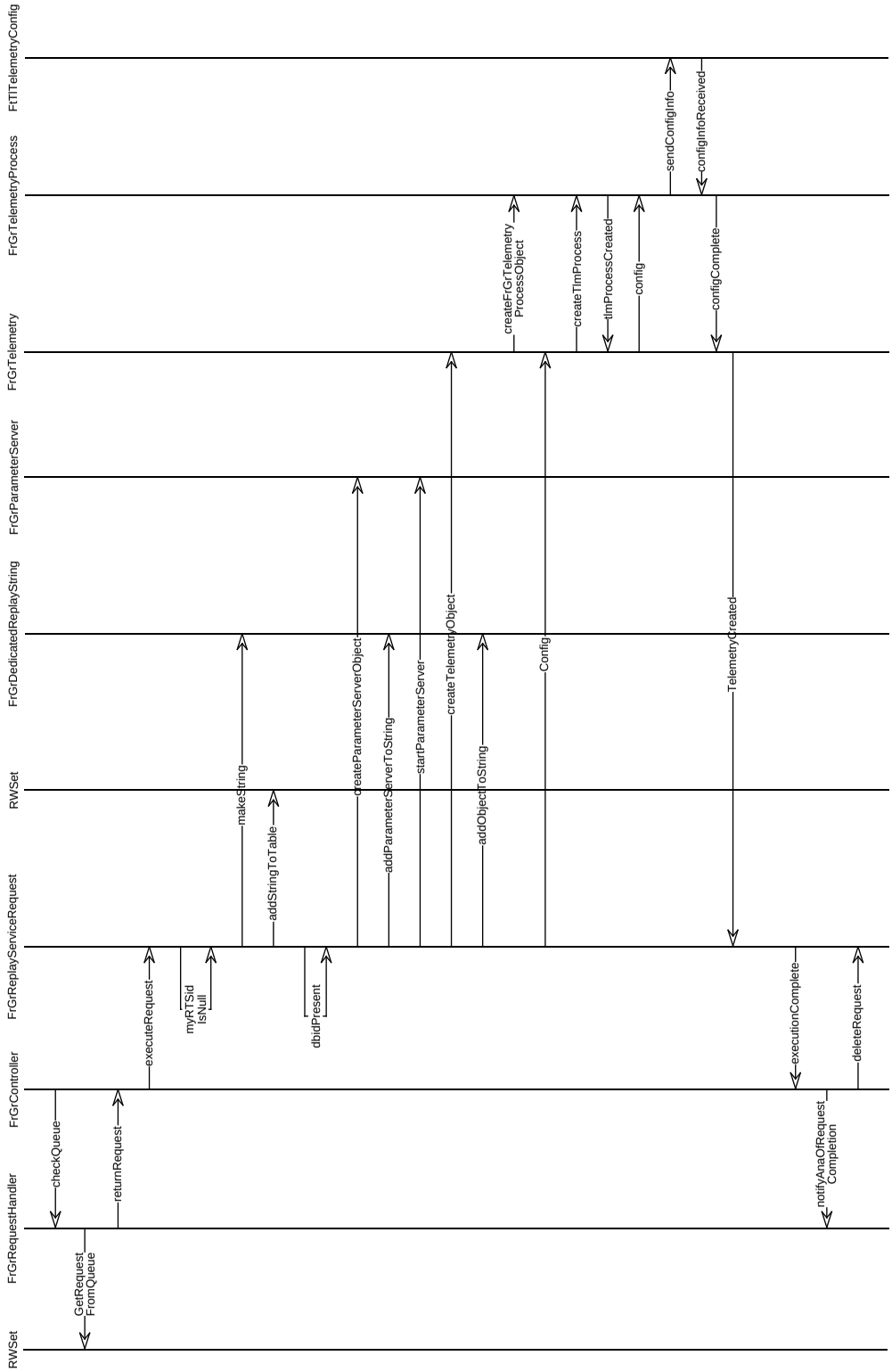


Figure 3.2.4.12.4-1 Request for Dedicated Replay Telemetry Arrives on the Workstation from Analysis Event Trace

3.2.4.13 Request for a String Failover Arrives on the Workstation Scenario

3.2.4.13.1 Request for a String Failover Arrives on the Workstation Abstract

The purpose of the Request for a String Failover Arrives on the Workstation scenario is to describe how the workstation RMS acts upon user request for a string failover.

3.2.4.13.2 Request for a String Failover Arrives on the Workstation Summary Information

Interfaces:

FOS User Interface Subsystem

Stimulus:

The user, wishing to failover a string, sends a StringFailoverRequest to RMS, via the user interface.

Desired Response:

The RMS software will forward the Request to the RTS where the failed string resides in order for it to be deactivated. Once the failed string has been deactivated, the Request is forwarded to the RTS where the active string resides in order for it to be activated. As a result, the failed string has been failed over to a backup string.

Pre-Conditions:

A backup string has already been created.

The user sending the Failover Request has Ground Control Authority on the failed and backup strings.

Post-Conditions:

The failed string has been switched to inactive. Commands cannot be sent to the spacecraft via the failed string. Telemetry data, NASCOM blocks, and CODA reports are not being archived by the failed string and GCMRs cannot be sent to NCC via the RCM software.

The backup string has been switched to an active string and all processing previously done via the failed string is done by the new active string.

3.2.4.13.3 Scenario Description

The Controller checks the queue of the FrGrRequestHandler and a FrGrStringFailoverRequest is returned. The Controller invokes the StringFailoverRequest object's Execute operation. A string search is performed and the failed string is found. The UserId and User Workstation ID is ensured to be valid Ground Controller IDs. The myActionFlag attribute is set to Deactivate and the FrGrStringFailoverRequest is passed to the RTS RMS for processing.

Once the RTS RMS has finished processing the Request, another string search is performed and the backup string is found. The Request's Action Flag is set to "Activate" and sent to the RTS where the backup string resides. Once the RTS RMS has finished processing the Request, FUI is notified and the FrGrStringFailoverRequest object is deleted.

3.2.4.13.4State Transition Description

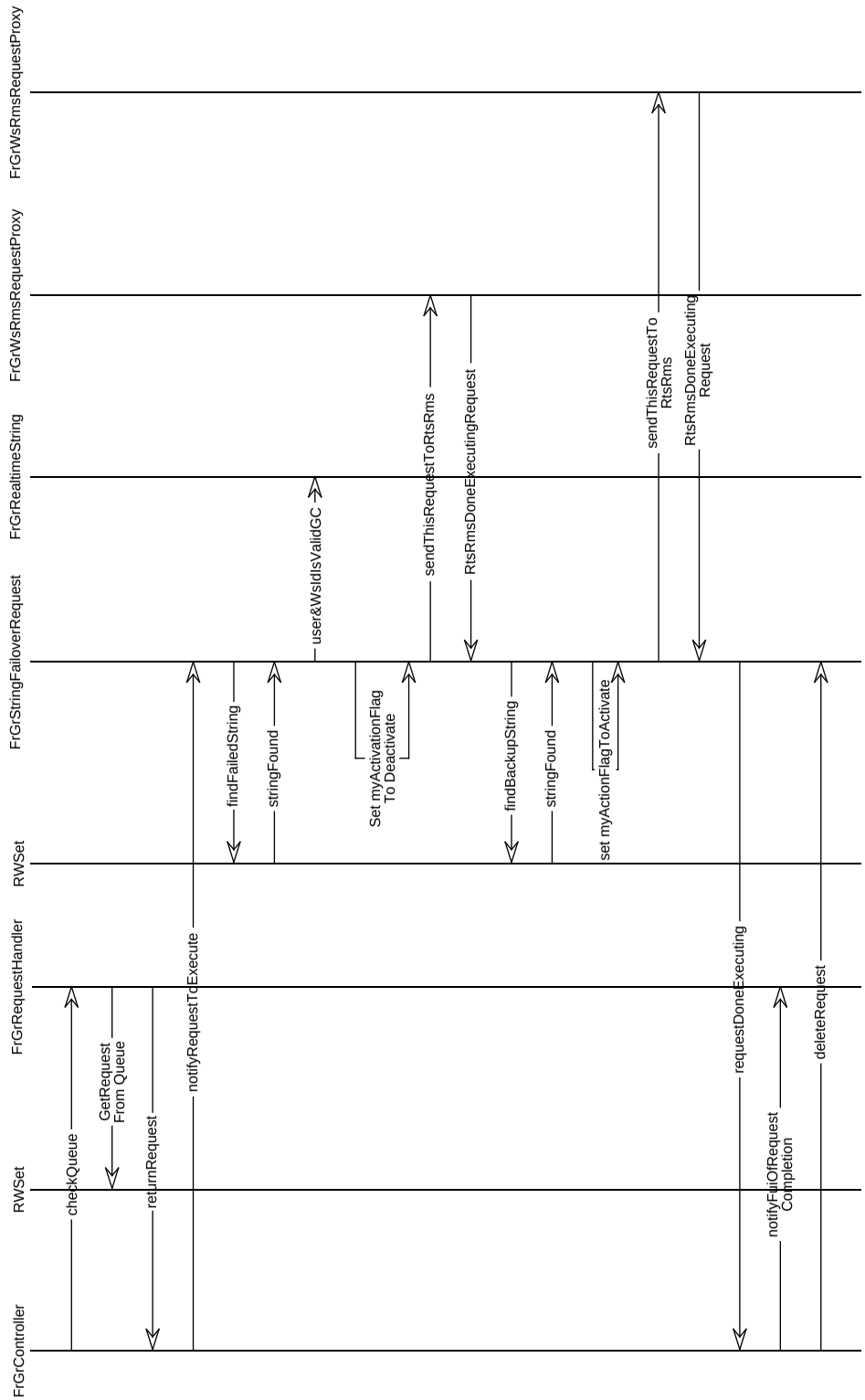


Figure 3.2.4.13.4-1. Request for a String Failover Arrives on the Workstation Event Trace

3.2.4.14 Request for String Deactivation Arrives on the Real-Time Server Scenario

3.2.4.14.1 Request for String Deactivation Arrives on the Real-Time Server Abstract

The purpose of the Request for String Deactivation Arrives on the Real-Time Server scenario is to describe how the RTS RMS acts upon a request for string deactivation. String deactivation involves the change of the logical string's operational state from Active to Inactive.

3.2.4.14.2 Request for String Deactivation Arrives on the Real-Time Server Summary Information

Interfaces:

Command Subsystem

Real-Time Contact Management Subsystem

Telemetry Subsystem

FUI Ground Script Controller Subsystem

Stimulus:

The workstation RMS sends a FrGrStringFailoverRequest object to the RTS RMS for string deactivation.

Desired Response:

The RMS software will deactivate a failed string by notifying Command, RCM, Telemetry, and the Ground Script Controller software of its new state.

Pre-Conditions:

The string that the RTS RMS is attempting to deactivate will need to already exist.

Post-Conditions:

The Command, RCM, Telemetry, and Ground Script Controller software will be notified of its inactive state.

3.2.4.14.3 Scenario Description

The Controller checks the queue of the FrGrRmsWsRmsIF and a FrGrStringFailoverRequest is returned. The Controller invokes the FrGrStringFailoverRequest object's Execute operation. The Action Flag is checked and found to have a value of "Deactivate." This indicates to the Request object that it needs to deactivate a string rather than activate it. A string search is performed and the failed string is found. After changing the string's state, a TableUpdateRequest object is created and multicasted to each WS RMS in order for the WS RMS String Table to be updated. The FrGrCommand object is retrieved from the string. In Figure 3.2.4.15.4-2, the FrGrCommand object's state is changed. The corresponding command processes are notified of their new state via the FrGrCommandProcess objects. In Figure 3.2.4.14.4-1, the Request object is notified that the Command state has been changed and the FrGrRTContact object is retrieved from the string. In Figure 3.2.4.15.4-4, the FrGrRTContact object's state is changed. The new state is sent to the RCM processes via the FrGrRcmProcess objects. In Figure 3.2.4.14.4-1, the Request object is notified that the RTContact state has been changed and the FrGrTelemetry object is retrieved from

the string. In Figure 3.2.4.15.4-3, the FrGrTelemetry object's state is changed. The new state is sent to the telemetry processes via the FrGrTelemetryProcess objects. In Figure 3.2.4.14.4-1, the Request object is notified that the Telemetry state has been changed and the FrGrGroundScriptController object is retrieved from the string. In Figure 3.2.4.15.4-4, the FrGrGroundScriptController object's state is changed and the new state is sent to the Ground Script Controller process. In Figure 3.2.4.14.4-1, the Request object is notified that the Ground Script Controller state has been changed. The WS RMS is notified that the Request has been processed, via the Controller, and the FrGrStringFailoverRequest is deleted.

3.2.4.14.4State Transition Description

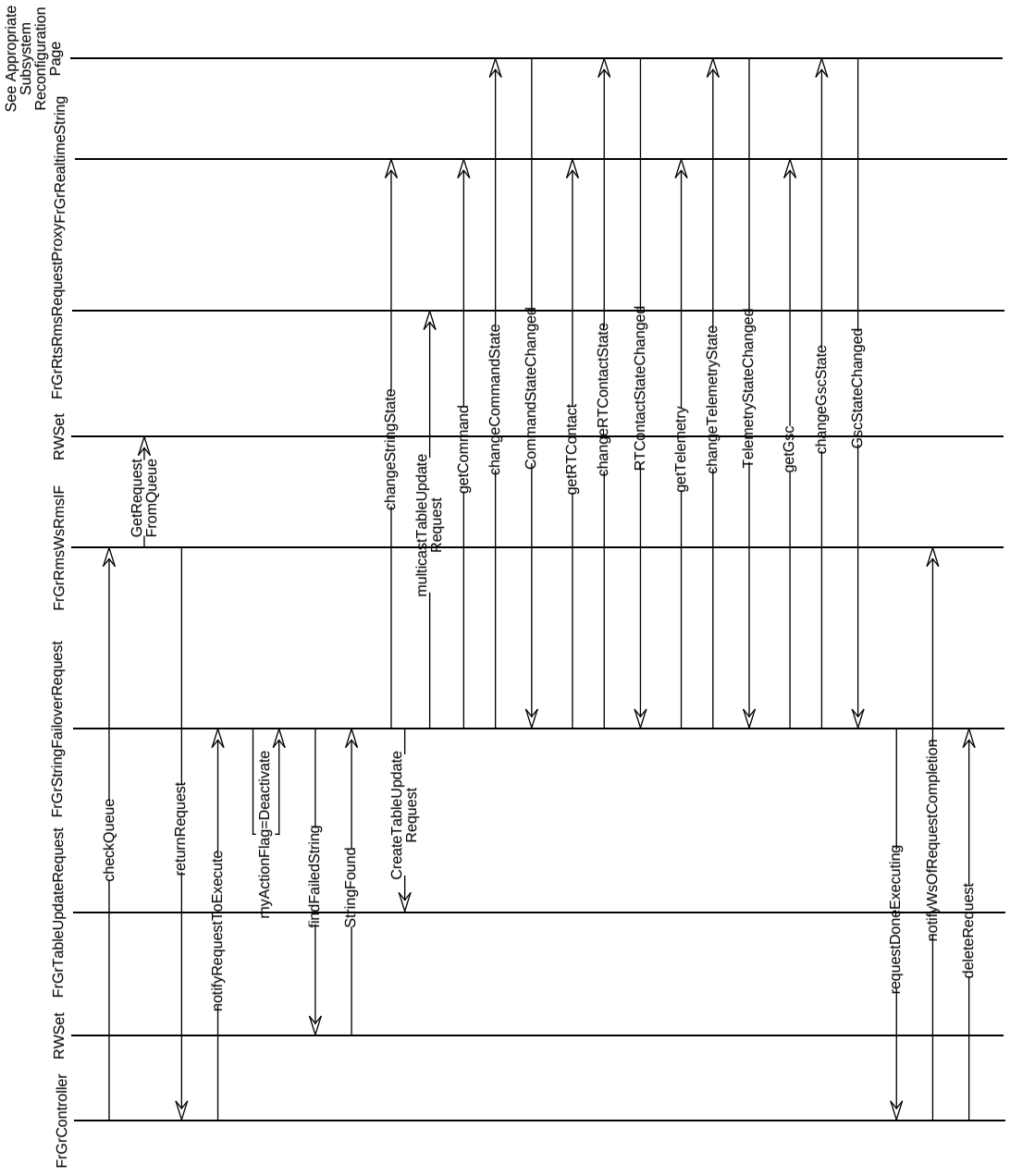


Figure 3.2.4.14.4-1. Request for String Deactivation Arrives on the Real-Time Server Event Trace

3.2.4.15 Request for String Activation Arrives on the Real-Time Server Scenario

3.2.4.15.1 Request for String Activation Arrives on the Real-Time Server Abstract

The purpose of the Request for String Activation Arrives on the Real-Time Server scenario is to describe how the RTS RMS acts upon a request to activate a logical string. String activation involves the change of the logical string's operational state from Backup to Active.

3.2.4.15.2 Request for String Activation Arrives on the Real-Time Server Summary Information

Interfaces:

Command Subsystem

Real-Time Contact Management Subsystem

Telemetry Subsystem

FUI Ground Script Controller Subsystem

Stimulus:

The workstation RMS sends a FrGrStringFailoverRequest object to the RTS RMS for string activation.

Desired Response:

The RMS software will activate a backup string by notifying Command, RCM, Telemetry, and the Ground Script Controller software of its new state.

Pre-Conditions:

The string that the RTS RMS is attempting to activate will need to already exist.

Post-Conditions:

The Command, RCM, Telemetry, and Ground Script Controller software will be notified of its active state.

3.2.4.15.3 Scenario Description

The Controller checks the queue of the FrGrRmsWsRmsIF and a FrGrStringFailoverRequest is returned. The Controller invokes the FrGrStringFailoverRequest object's Execute operation. The Action Flag is checked and found to not have a value of "Deactivate." This indicates to the Request object that it needs to activate a string rather than deactivate it. A string search is performed and the backup string is found. The string's state is changed and the myFailedRTSid is found to not have a value of "FAILED." Therefore, a message does not have to be multicasted to the WS RMS's to remove the failed string from their string tables. The RTS that the failed string resides on is capable of updating the string table. The backup string's ActiveStringID attribute is changed to NULL as a result of the string becoming active rather than backing up the string identified by the ActiveStringID. A TableUpdateRequest object is created and multicasted to each WS RMS in order for the WS RMS String Table to be updated. The FrGrTelemetry object is retrieved from the string. In Figure 3.2.4.15.4-3, the FrGrTelemetry object's state is changed. The new state is sent to the telemetry processes via the FrGrTelemetryProcess objects. In Figure 3.2.4.15.4-1, the

Request object is notified that the Telemetry state has been changed and the FrGrCommand object is retrieved from the string. In Figure 3.2.4.15.4-2, the FrGrCommand object's state is changed. The corresponding command processes are notified of their new state via the FrGrCommandProcess objects. In Figure 3.2.4.15.4-1, the Request object is notified that the Command state has been changed and the FrGrRTContact object is retrieved from the string. In Figure 3.2.4.15.4-4, the FrGrRTContact object's state is changed. The new state is sent to the RCM processes via the FrGrRcmProcess objects. In Figure 3.2.4.15.4-1, the Request object is notified that the RTContact state has been changed and the FrGrGroundScriptController object is retrieved from the string. In Figure 3.2.4.15.4-4, the FrGrGroundScriptController object's state is changed and the new state is sent to the Ground Script Controller process. In Figure 3.2.4.15.4-1, the Request object is notified that the Ground Script Controller state has been changed. The WS RMS is notified that the Request has been processed, via the Controller, and the FrGrStringFailoverRequest is deleted.

3.2.4.15.4State Transition Description

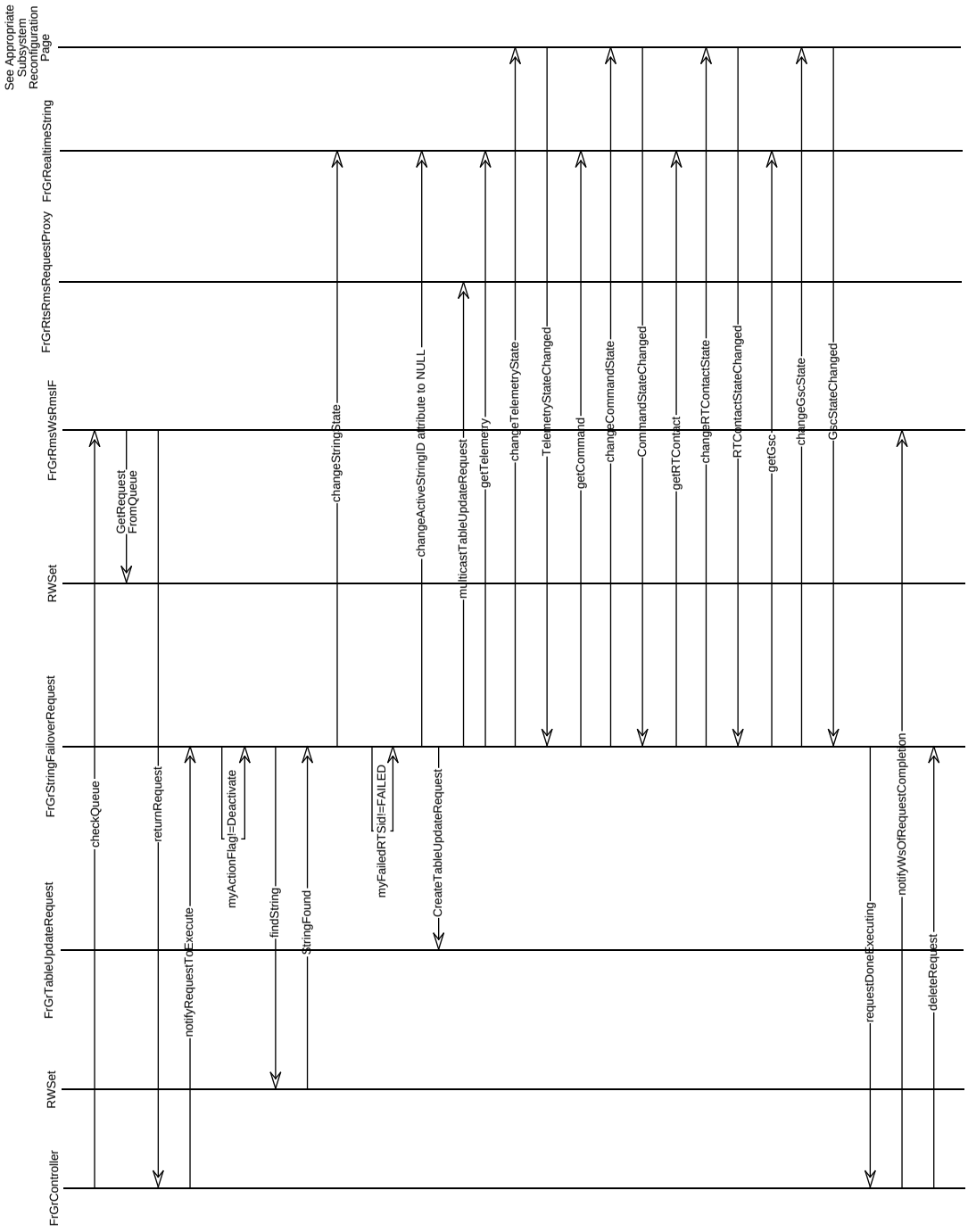


Figure 3.2.4.15.4-1. Request for String Activation Arrives on the Real-Time Server Event Trace

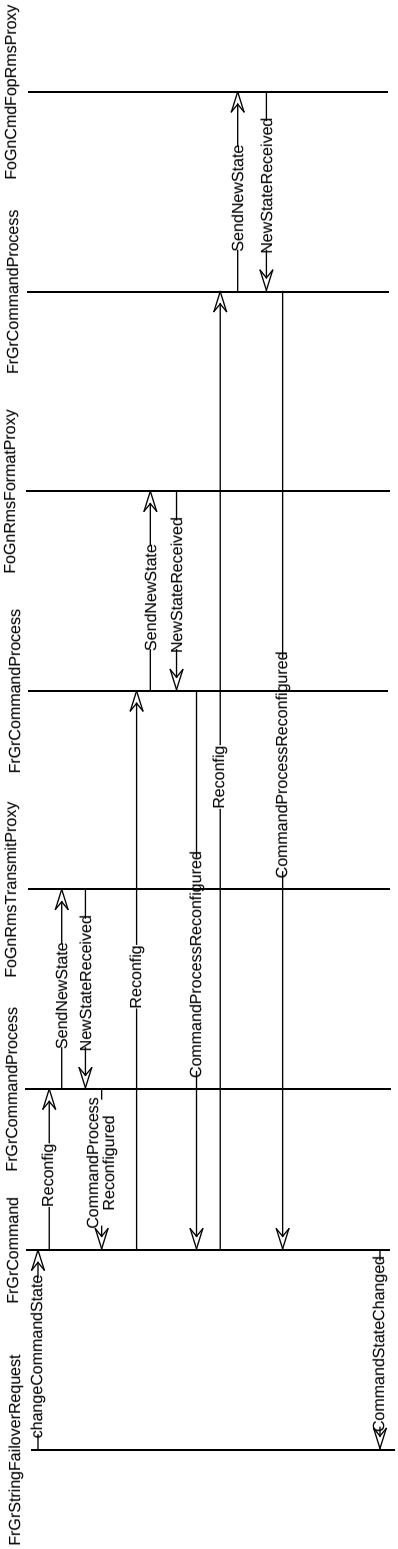


Figure 3.2.4.15.4-2. Command State Change Event Trace

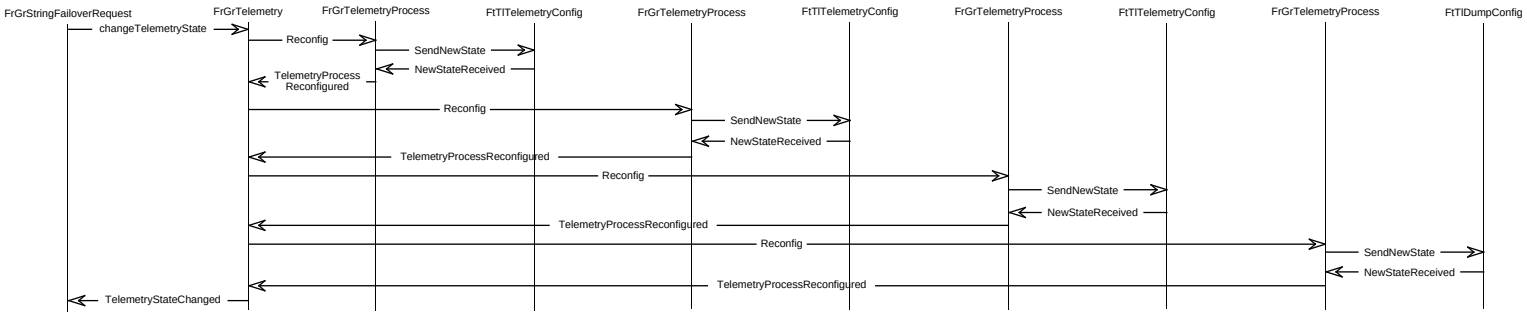


Figure 3.2.4.15.4-3. Telemetry State Change Event Trace

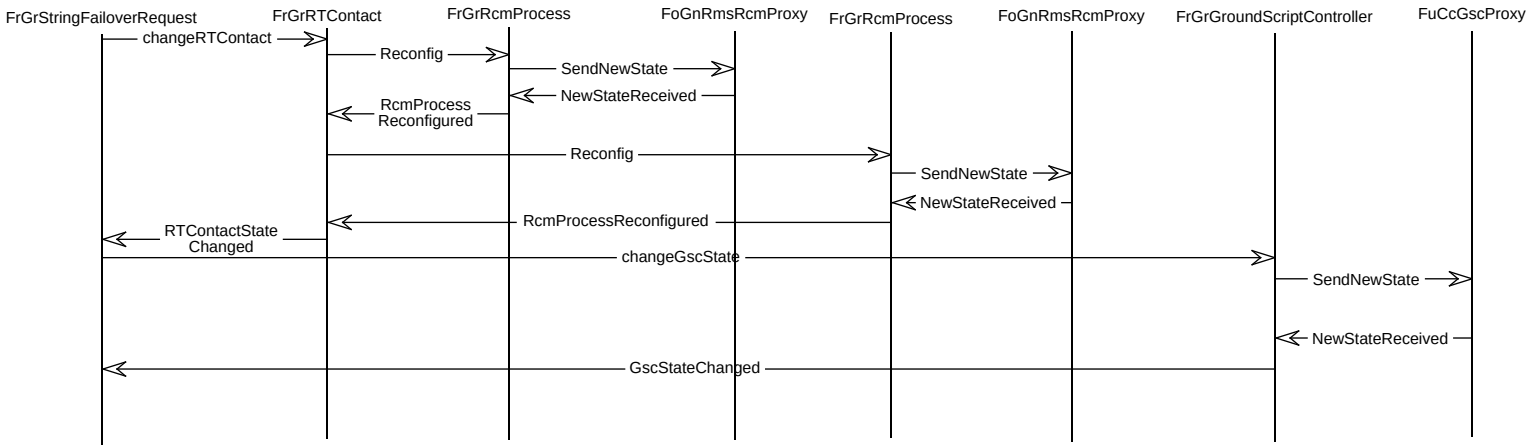


Figure 3.2.4.15.4-4. Real-Time Contact Management and Ground Script Controller

3.2.5 RMS String Manager Component Data Dictionary

FrGrAdjustLimitRequest

class **FrGrAdjustLimitRequest**

This class is responsible for containing all functionality necessary for processing the limit adjust request.

Base Classes

public **FrGrRequest**

Public Construction

FrGrAdjustLimitRequest(const FrGrAdjustLimitRequest&)

This routine creates a duplicate of this class.

FrGrAdjustLimitRequest()

This routine creates an instance of this class.

~FrGrAdjustLimitRequest()

This routine deletes an instance of this class.

Public Functions

EcTInt **configTlm**()

This routine finds the appropriate telemetry object i and

notifies it of the configuration change.

EcTInt **execute**(FrGrController* Controller)

This routine is responsible for containing all functionality necessary for processing the request. It calls additional subroutines in order to accomplish this task.

EcTInt **sendConfigChange**(FrGrRmsWsRmsIF* PassedFrGrRmsWsRmsIF)

This routine iterates through the list of mirrored workstations and sends them the configuration change.

Private Data

EcTInt **myParameterId**

This member variable identifies the parameter identifier

of the parameter for which tht user has requested a limit setting change.

EcTInt **mySetId**

This member variable identifies the set of parameters to

be affected by the request.

RWCString **myTelemetryType**

This member variable indentifies the telemetry subsystem to be reconfigured. This could be HouseKeeping, Health and Safety, Standby or Diagnostic.

RWCString **myType**

This member variable identifies tht type of limit, i.e. boundary, rail or delta.

EcTDouble **myValue**

This member variable identifies the new limit value

FrGrBackupServiceRequest

class **FrGrBackupServiceRequest**

This class is responsible for containing all functionality necessary for processing the backup service request.

Base Classes

public **FrGrRequest**

Public Construction

FrGrBackupServiceRequest(const FrGrBackupServiceRequest&)

This routine creates a duplicate of this class.

FrGrBackupServiceRequest()

This routine creates an instance of this class.

~FrGrBackupServiceRequest()

This routine deletes an instance of this class.

Public Functions

EcTInt **execute**(FrGrController* Controller)

This routine is responsible for containing all functionality necessary for processing the request. It calls additional subroutines in order to accomplish this task.

Protected Functions

EcTInt **CreateGSC**(FrGrStrManResMonIF* PassedStrManResMonIF)

This routine creates a GroundScriptController object for FUI and sets the FUI attributes, it also adds the GroundScriptController object to the string resource list.

EcTInt **createCmd**(FrGrStrManResMonIF* PassedStrManResMonIF)

This routine is responsible for creation of the Command Object and notification of the Command Object that it needs to configure a Command Process. In addition, it notifies the Resource Monitor Task of the new Command Process.

EcTInt **createParamServer**(FrGrStrManResMonIF* PassedStrManResMonIF)

This routine is responsible for the creation of the Parameter Server Object.

EcTInt **createRcm**(FrGrStrManResMonIF* PassedStrManResMonIF)

This routine is responsible for creation of the RTContact Object and notification of the RTContact Object that it needs to configure a RCM Process. In addition, it notifies the Resource Monitor Task of the new RCM Process.

EcTInt **createTlm**(FrGrStrManResMonIF* PassedStrManResMonIF)

This routine is responsible for creation of the Telemetry Object and notification of the Telemetry Object that it needs to configure a particular telemetry process. In addition, it notifies the Resource Monitor Task of the new Telemetry Process.

FrGrString* **findString**(RWSet* PassedStringTable, int PassedStringID)

This routine finds a particular string in a string table based on the StringId.

FrGrString* **makeString**()

This routine makes a particular string based on attributes of the request.

Private Data

RWCString **myCaWsId**

This member variable identifies the workstation with Command Authority.

RWCString **myCaId**

This member variable identifies the person with Command Authority.

RWCString **myDiagnosticTlmConfigFilename**

This member variable identifies the configuration filename for the Diagnostic Telemetry process.

RWCString **myEoutMgrRcmConfigFilename**

This member variable identifies the configuration filename for the RCM EoutMgr process.

RWCString **myFopCmdConfigFilename**

This member variable identifies the configuration filename for the Command process.

RWCString **myFormatCmdConfigFilename**

This member variable identifies the configuration filename for the Format Command process.

RWCString **myGcId**

This member variable identifies the person with Ground Station priviledge.

RWCString **myGcWsId**

This member variable identifies the workstation with Ground Control priviledge.

RWCString **myHStlmConfigFilename**

This member variable identifies the configuration filename for the Health & Safety

Telemetry process.

RWCString **myHkTlmConfigFilename**

This member variable identifies the configuration filename for the Housekeeping Telemetry process.

RWCString **myMode**

This member variable identifies the logical string mode, i.e. operational, test or training.

RWCString **myNoutMgrRcmConfigFilename**

This member variable identifies the configuration filename for the RCM NoutMgr process.

EcTInt **myRTSid**

This member variable identifies the Real-Time Server on which this backup string is to be created.

RWCString **mySbTlmConfigFilename**

This member variable identifies the configuration filename for the Standby Telemetry process.

RWCString **myScid**

This member variable identifies the spacecraft.

EcTInt **myStringID**

This member variable identifies the logical string for which this backup string is to be created.

RWCString **myUplinkCmdConfigFilename**

This member variable identifies the configuration filename for

the Uplink Command process.

RWCString **myUserID**

This member variable identifies the user from whom this request originated.

RWCString **myWksID**

This member variable identifies the user station from which this request originated.

FrGrCommand

class **FrGrCommand**

A description of the class

Base Classes

public **FrGrSoftware**

Public Construction

FrGrCommand(const FrGrCommand&)

FrGrCommand(const FrGrCommand&)

This member function creates a duplicate of this class.

FrGrCommand()

This member fuunction is the default constructor for this class

~FrGrCommand()

This member function is the destructor for this class. It will call the FrGrCommand::Stop() operation.

Public Functions

EcTInt **ChangeState**(State)

This member function will change the myState attribute of this class as well as send the change to the CMD processes.

EcTInt **Config**(RWCString FormatFile, RWCString TransmitFile, RWCString FopFile)

Config(RWCString FormatFile, RWCString TransmitFile, FopFile)

This member function calls the MakeRmsAddress and the MakeDmsAddress subroutines before calling myCmdFormat->StartCmdProcess(), myCmdTransmit->StartCmdProcess() and myCmdFop->StartCmdProcess(). These operations will start and configure the corresponding CMD processes.

EcTInt **Config**(void)

This member function calls the MakeRmsAddress and the MakeDmsAddress subroutines before calling myCmdFormat->StartCmdProcess(), myCmdFop->StartCmdProcess() and myCmdTransmit->StartCmdProcess(). These operations will start and configure the corresponding CMD processes.

EcTInt **Reconfig**(RWCSting configParameter)

This member function will determine which CMD process will receive a configuration parameter and call either the myCmdFormat-> ReconfigCmdProcess(RWCString configParameter), myCmdFop-> ReconfigCmdProcess(RWCString configParameter) or myCmdTransmit-> ReconfigCmdProcess(RWCString configParameter).

EcTInt **Snap**(void)

This member function will notify the Format, Transmit and FOP processes to take a configuration snapshot of themselves and update their mySnapFilename attribute.

EcTInt **Stop**(void)

This member function will call myCmdFormat->StopCmdProcess(), myCmdFOP-

>StopCmdProcess and myCmdTransmit->StopCmdProcess().

Protected Functions

EcTInt **MakeDmsAddress**(void)

This member function will set the myRmsAddress attribute which will be used to start the CMD processes.

EcTInt **MakeRmsAddress**(void)

This member function will set the myDmsAddress attribute which will be used to configure the CMD processes.

Private Data

FrGrCommandProcess* **myCmdFop**

This member variable points to the Fop object.

FrGrCommandProcess* **myCmdFormat**

This member variable points to the Format object.

FrGrCommandProcess* **myCmdTransmit**

This member variable points to the Transmit object.

Address* **myDmsAddress**

This member variable identifies the address of the DMS process that the CMD processes will communicate with.

FrGrCommandPriviledgeRequest

class **FrGrCommandPriviledgeRequest**

This routine is responsible for containing all functionality necessary for processing the command priviledge request. It calls additional subroutines in order to accomplish this task

Public Construction

FrGrCommandPriviledgeRequest(const FrGrCommandPriviledgeRequest&)

This routine creates a duplicate of this class.

FrGrCommandPriviledgeRequest()

This routine creates an instance of this class.

~FrGrCommandPriviledgeRequest()

This routine deletes an instance of this class.

FrGrCommandProcess

class **FrGrCommandProcess**

This class us used to create, destroy, configure and reconfigure any of the potential command processes, i.e. Format, Transmit or Frame Operations Procedure (FOP). It will also take a

configuration snapshot of any of the aforementioned processes.

Base Classes

public **FrGrCommand**

Public Construction

FrGrCommandProcess(const FrGrCommandProcess&)

This member function is a "copy constructor", it creates a duplicate of this class.

FrGrCommandProcess(RWCString myDbid, RWCString myScid, myState, Address* myRmsAddress)

This member function is the default constructor for this class.

~FrGrCommandProcess()

This member function is the destructor for this class. It will call the FrGrCommandProcess::StopCmdProcess() member function.

Public Functions

EcTInt **ConfigCmdProcess**()

This member function sends configuration information to the FrGrCommandProcess process after it has been started.

EcTInt **ReconfigCmdProcess**(RWCString configParameter)

This member function sends a reconfiguration parameter to the specified command process.

EcTInt **SnapCmdProcess**(void)

This member function requests a configuration snapshot of the FrGrCommandProcess process and updates the mySnapFilename attribute.

EcTInt **SnapConfigCmdProcess**(RWCString ConfigFile)

This member function sends snapshot configuration information to a command process after it has been started.

EcTInt **StartCmdProcess**()

This member function will create a command process of one of the following command types: Format, Transmit or FOP.

Protected Functions

EcTInt **StopCmdProcess**(void)

This member function will terminate a FrGrCommandProcess process.

Private Data

FoGnCmdFopRmsProxy* **myCmdFopProxy**

This member variable points to the FoGnCmdFopRmsProxy.

EcTInt **myCmdPid**

This member variable identifies the command processes, Pid

RWCString **myDbId**

This member variable is the database identifier which is to be used by this process for the extraction of database information.

FoGnRmsFormatProxy* **myFormatProxy**

This member variable points to the FoGnRmsFormatProxy.

FrGrCommand* **myFrGrCommand**

This member variable points to the FrGrCommand object that this object is part of.

Address* **myRmsAddress**

This member variable points to the address of the RMS process.

RWCString **myScId**

This member variable is the identifier of the space craft of which this process is supporting.

RWCString **mySnapFilename**

This member variable identifies a snapshot configuration filename.

FoGnRmsTransmitProxy* **myTransmitProxy**

This member variable points to the FoGnRmsTransmitProxy.

FrGrController

class **FrGrController**

This class serves as the controller or coordinator for the Resource Management Subsystem. Among its duties is the coordination of creation, deletion, and updates to all RMS-sponsored software that is part of a logical string. This is done through user directive and resource status changes.

Public Construction

FrGrController(const FrGrController&)

FrGrController(const FrGrController&)

This member function creates a duplicate of this class.

FrGrController()This member function is the default constructor for this class.

~FrGrController()

This member function is the destructor for this class.

Public Functions

int **initialize**()

This routine initializes the RMS based on the type of host it is running on.

int **receiveRequest**(FrGrRequest* receivedRequest)

This member function notifies a request that it is time to execute.

void **run**()

This routine notifies the FrGrRmsWsRmsIF object to begin its runs state.

Protected Functions

EcTInt **MakeFdDsFileAccessor**()

This routine creates the proxy for accessing DMS files.

EcTInt **MakeFrGrRtsRmsRequestProxy**()

This routine create a FrGrRtsRmsRequestProxy object in order for the RTS RMS to communicate with the WS RMS's.

int **exDefConDirs**()

This routine iterates through the set of Default Configuration Directives and executes them.

int **loadDefConInfo**()

This routine loads the myDefConInfo object from the file names by the myDefConFile attribute.

int **makeDefConInfo**()

This routine sets the myDefConInfo pointer to point to a RWSet. This will be used to store the Default Configuration Requests.

EcTInt **makeFdEvEventLoggerobject**()

makeFdEvEventLogger

This routine creates a FdEvEventLoggerIF object that will be used to communicate with DMS to log events.

int **makeFoGnCsmsIFobject**()

This routine creates the FoGnCsmsIFobject that will be used to communicate with CSMS for user and workstation authorization.

int **makeFoPsClientIF**()

This routine spawns a Parameter Server process and creates a FoGnRmsPsIF

int **makeFrGrRequestHandler**()

This routine creates a FrGrRequestHandler object that will be used to receive Requests.

int **makeFrGrRmsWsRmsIFobject**()

This routine creates the FrGrRmsWsRmsIFobject that will be used to communicate with the WS RMS or the RTS RMS.

int **makeFrGrWsRmsRequestProxies**()

This routine creates a FrGrWsRmsRequestProxy object for each RTS and places that IF object into a RWSet.

int **makeFrGrWsRmsRequestProxy**(int RMSnodeID)

This routine creates a FrGrWsRmsRequestProxy object for the WS RMS to communicate with the RTS RMS.

int **makeStrManResMonProxy**()

This routine creates a FrGrStrManResMonProxy object that will be used to communicate with the Resource Monitor process.

int **makeStringTable**()

This routine sets the myStringTable pointer to point to a RWSet.

int **queryRTSstrings**()

This routine requests the string table from each RTS RMS in order for the WS RMS can construct a comprehensive string table.

int **reqDefConFile**()

This routine sets the myDefConFile attribute and sends the request for the file to the FoGnRmsDmsIF object.

int **setRole**()

This routine determines the type of host that the RMS is running on and sets the myHost attribute.

Private Data

FrGrWsRmsRequestProxy* **myCurrentFrGrWsRmsRequestProxy**

This attribute points to a particular workstation RMS/RTS RMS interface object, depending on which RTS RMS the workstation RMS is communicating with at a given time.

RWCString **myDefConFile**

This attribute identifies the Default Configuration File to be used at initialization.

RWSet* **myDefConInfo**

This attribute points to the collection of Default Configuration Directives.

FdEvEventLogger* **myFdEvEventLogger**

This attribute points to the RMS/DMS Event Logger interface object.

FoDsFileAccessor* **myFileAccessor**

This attribute points to the FileAccessor proxy.

FoGnCsmsIF* **myFoGnCsmsIF**

This attribute points to the CSMS interface object.

FoPsClientIF* **myFoPsClientIF**

This attribute points to the Parameter Server Client IF.

FrGrRequestHandler* **myFrGrRequestHandler**

This attribute points to the Request Handler object.

FrGrRmsWsRmsIF* **myFrGrRmsWsRmsIF**

This attribute points to the receiver object for WS RMS to RTS RMS communication.

FrGrStrManResMonProxy* **myFrGrStrManResMonProxy**

This attribute points to the interface object responsible for communication between the RMS String Manager process and the RMS Resource Monitor process.

RWSet* **myFrGrWsRmsRequestProxySet**

This attribute points to a collection of interface objects responsible for letting a workstation RMS communicate with each RTS RMS.

RWCString **myHost**

This attribute identifies the host machine where this instance of the RMS process is running.

RWCString **myOperationalDB**

This attribute identifies the operational database that this RMS process is using.

FrGrRtsRmsRequestProxy* **myRtsRmsRequestProxy**

This attribute points to the RtsRmsRequestProxy object.

RWSet* **myStringTable**

This attribute points to the RMS String Table.

FrGrDataArchiver

class **FrGrDataArchiver**

This class is used to create, destroy, and configure, the Data Archiver Process.

Public Construction

FrGrDataArchiver(const FrGrDataArchiver&)

FrGrDataArchiver(const FrGrDataArchiver&)

This member function creates a duplicate of this class.

FrGrDataArchiver()

This member function is the default constructor for this class.

~FrGrDataArchiver()

This member function is the destructor for this class.

Public Functions

EcTInt **Config**()

This member function sends configuration information to the DataArchiver process after it has been started.

EcTInt **Start**()

This member function will create the Data Archiver process.

EcTInt **Stop**(void)

This member function will terminate a DataArchiver process.

Private Data

RWCString **myDbId**

This member variable identifies the Database that the data archiver process will use.

EcTInt **myPid**

This member variable identifies the Process ID that is associated with the Data Archiver process.

FdCfRMSConfigProxy* **myRMSConfigProxy**

This member variable points to the RMSConfigProxy.

Address* **myRmsAddress**

This member variable identifies the RMS process that the DataArchiver process is started by.

RWCString **myScId**

This member variable identifies the spacecraft that the DataArchiver process is associated with. It will be used as configuration data.

FrGrGroundControlPriviledgeRequest

class **FrGrGroundControlPriviledgeRequest**

This class is responsible for containing all functionality necessary for processing the ground control priviledge request. It calls additional subroutines in order to accomplish this task.

Public Construction

FrGrGroundControlPriviledgeRequest(const FrGrGroundControlPriviledgeRequest&)

This routine creates a duplicate of this class.

FrGrGroundControlPriviledgeRequest()

This routine creates an instance of this class.

~FrGrGroundControlPriviledgeRequest()

This routine deletes an instance of this class.

FrGrGroundControlRequest

class **FrGrGroundControlRequest**

This class finds a particular active or backup string in a string table based on the StringId.

Base Classes

public **FrGrRequest**

Public Construction

FrGrGroundControlRequest(const FrGrGroundControlRequest&)

This routine creates a duplicate of this class.

FrGrGroundControlRequest()

This routine creates an instance of this class.

~FrGrGroundControlRequest()

This routine deletes an instance of this class.

Private Data

EcTInt **mySendFlag**

This member variable identifies if the object needs to be forwarded to the RTS RMS.

EcTInt **myStringID**

This member variable identifies the logical string to which this request refers.

RWCString **myUserID**

This member variable identifies the user from whom this request originated.

RWCString **myWksID**

This member variable identifies the user station form which this request originated.

FrGrGroundScriptController

class **FrGrGroundScriptController**

This class is used to create, destroy, configure, and reconfigure the Ground Script Controller process on the RTS.

Base Classes

public **FrGrSoftware**

Public Construction

FrGrGroundScriptController(const FrGrGroundScriptController&)

FrGrGroundScriptController(const FrGrGroundScriptController&)

This member function creates a duplicate of this class.

FrGrGroundScriptController()

This member function is the default constructor for this class.

~FrGrGroundScriptController

This member function is the destructor for this class. It will call the FrGrGroundScriptController::Stop() operation.

Public Functions

EcTInt **Config**()

This member function sends configuration information to the GroundScriptController process after it has been started.

EcTInt **Reconfig**(EcTInt configparam)

Reconfig(EcTInt)

This member function will change a GroundScriptController configuration parameter that is an EcTInt type.

EcTInt **Reconfig**(RWCString configparam)

Reconfig(RWCString)

This member function will change a GroundScriptController configuration parameter that is a string type.

EcTInt **Start**()

This member function will create the GroundScriptController process and pass it the configuration data.

Public Data

~FrGrGroundScriptController

This member function is the destructor for this class. It will call the FrGrGroundScriptController::Stop() operation.

Protected Functions

EcTInt **Stop**(void)

This member function will terminate a GroundScriptController process.

Private Data

Address* **myCmdAddress**

This member variable identifies the command process that the GroundScriptController process is associated with. It will be used as configuration data.

FuCcGscProxy* **myGscProxy**

This member variable points to the GroundScriptController Proxy.

EcTInt **myRtsId**

This member variable identifies the RTS the GroundScriptController process is running on. It will be used as configuration data.

EcTInt **myStringId**

This member variable identifies the String that the GroundScriptController process is associated with. It will be used as configuration data.

Address* **myTlmAddress**

This member variable identifies the telemetry process that the GroundScriptController process is associated with. It will be used as configuration data.

FrGrParameterServer

class **FrGrParameterServer**

This class is used to create and destruct an instance of the Parameter Server.

Public Construction

FrGrParameterServer(const FrGrParameterServer&)

FrGrParameterServer(const FrGrParameterServer&)

This member function creates a duplicate of this class.

FrGrParameterServer()

This member function is the default constructor for this class.

~FrGrParameterServer()

This member function is the destructor for this class.

Public Functions

EcTInt **Start**()

Start()

This member function will start a Parameter Server process.

EcTVoid **Stop**()

Stop()

This member function will stop a Parameter Server process.

Private Data

EcTInt **myPsPid**

This member variable is the process id of the this Parameter Server.

FrGrPriviledgeRequest

class **FrGrPriviledgeRequest**

This routine finds a particular active or backup string in a string table based on the StringID.

Base Classes

public **FrGrRequest**

Public Construction

FrGrPriviledgeRequest(const FrGrPriviledgeRequest&)

This routine creates a duplicate of this class.

FrGrPriviledgeRequest()

This routine creates an instance of this class.

~FrGrPriviledgeRequest()

This routine deletes an instance of this class.

Private Data

EcTInt **myStringID**

This member variable identifies the logical string to which this request refers.

RWCString **myUserID**

This member variable identifies the users from whom this request originated.

RWCString **myWksID**

This member variable identifies the user station from which this request originated.

FrGrRTContact

class **FrGrRTContact**

This class is used to create, destroy, configure, and reconfigure the RCM subsystem. In addition, it will take a configuration snapshot of the RCM subsystem.

Base Classes

public **FrGrSoftware**

Public Construction

FrGrRTContact(const FrGrRTContact&)

FrGrRTContact(const FrGrRTContact&)

This member function creates a duplicate of this class.

FrGrRTContact()

This member function is the default constructor for this class.

~FrGrRTContact()

This member function is the destructor for this class. It will call the FrGrRTContact::Stop() operation.

Public Functions

EcTInt **ChangeState**(State)

This member function will change the myState attribute of this class as well as send the change to the RCM processes.

EcTInt **Config**(RWCString EoutFile, RWCString NoutFile)

Config(RWCString EoutFile, RWCString NoutFile)

This member function calls the MakeRmsAddress subroutines before calling the StartRcmProcess() for the NoutMgr or EoutMgr process These operations will start and configure the corresponding RCM processes.

EcTInt **Config**()

This member function calls the MakeRmsAddress subroutines before calling the StartRcmProcess() for the NoutMgr, NinMgr,

EoutMgr or EinMgr process.

EcTInt **Reconfig**(RWCString configParam)

This member function will determine which RCM process will receive a configuration parameter and call either the ReconfigRcmProcess() for

either the NoutMgr, NinMgr, EoutMgr or EinMgr process.

EcTInt **Snap**(void)

This member function will notify each RCM process to take a configuration snapshot of their corresponding processes and update their mySnapFilename attribute.

EcTInt **Stop**(void)

This member function will call the StopRcmProcess() member

function for each RCM process.

Protected Functions

EctInt **MakeRmsAddress**()

This member function will set the myRmsAddress attribute which will be used to start the RCM processes.

Private Data

FrGrRcmProcess* **myEoutMgr**

This member variable points to the EDOS output object.

FrGrRcmProcess* **myNoutMgr**

This member variable points to the NCC output object.

FrGrRcmProcess

class **FrGrRcmProcess**

This class is used to create, destroy, configure, and reconfigure an RCM process. In addition, it will take a configuration snapshot of the RCM process.

Public Construction

FrGrRcmProcess(const FrGrRcmProcess&)

FrGrRcmProcess(const FrGrRcmProcess&)

This member function creates a duplicate of this class.

FrGrRcmProcess(RWCString myDbid, RWCString myScid, myState, Address* myRmsAddress)

This member function is the default constructor for this class.

~FrGrRcmProcess()

This member function is the destructor for this class.

Public Functions

EcTInt **ConfigRcmProcess**()

This member function sends configuration information to the Rcm process after it has been started.

EcTInt **ReconfigRcmProcess**(RWCString configParam)

This member function sends a reconfiguration parameter to the Rcm process.

EcTInt **SnapConfigRcmProcess**(RWCString ConfigFile)

This member function sends snapshot configuration information to the Rcm process after it has been started.

EcTInt **SnapRcmProcess**()

This member function requests a configuration snapshot of the Rcm process and updates the mySnapFilename attribute.

EcTInt **StartRcmProcess**()

This member function will create the Rcm process.

Protected Functions

EcTInt **StopRcmProcess**()

This member function will terminate an Rcm process.

Private Data

FrGrDataArchiver* **myDataArchiver**

This member variable points to the associated FrGrDataArchiver object.

RWCString **myDbId**

This member variable is the database identifier which is to be used by this process for the extraction of database info.

FrGrRTContact* **myFrGrRTContact**

This member variable points to the FrGrRTContact object that this object is a part of.

RWCString **myProcessName**

This member variable identifies the Process Name. This can be the EoutMgr or the NoutMgr.

EcTInt **myRcmInPid**

This member variable identifies the Process ID of the process that is started by either NoutMgr or EoutMgr.

EcTInt **myRcmPid**

This member variable identifies the process Pid

Address* **myRmsAddress**

This member variable points to the address of the Rms process.

FoGnRmsRcmProxy* **myRmsRcmProxy**

This member variable points to the RMS/RCM proxy.

RWCString **myScId**

This member variable is the identifier of the space craft of which this process is supporting.

RWCString **mySnapFilename**

This member variable identifies a snapshot configuration filename.

FrGrRealtimeServiceRequest

class **FrGrRealtimeServiceRequest**

_FrGrRealtimeServiceRequest_h_

This class is responsible for processing a Realtime Service Request.

Base Classes

private **FrGrServiceRequest**

Public Construction

FrGrRealtimeServiceRequest(const FrGrRealtimeServiceRequest&)

This routine creates a duplicate of this class.

FrGrRealtimeServiceRequest()

This routine creates an instance of this class.

~FrGrRealtimeServiceRequest()

This routine deletes an instance of this class.

Public Functions

EcTInt **execute**(FrGrController* Controller)

This routine is responsible for containing all functionality necessary for processing the request. It calls additional subroutines in order to accomplish this task.

Protected Functions

EcTInt **createGSC**()

This routine is responsible for creation of the Ground Script Controller Object and notification of the Ground Script Controller Object that it needs to configure a particular Ground Script Controller process.

EcTInt **createParamServer**()

This routine is responsible for the creation of the Parameter Server Object.

EcTInt **createRCM**(FrGrStrManResMonIF* PassedStrManResMonIF)

createRcm

This routine is responsible for creation of the RTContact Object and notification of the RTContact Object that it needs to configure a RCM Process. In addition, it notifies the Resource Monitor Task of the new RCM Process.

FrGrString* **findString**(RWSet* PassedStringTable)

This routine finds a particular string in a string table based on the request attributes.

FrGrString* **makeString**()

This routine makes a particular string based on attributes of the request.

Private Data

RWCString **myMode**

This member variable is the identifier of the string's mode. This can be operational, test, or training.

RWCString **myMonitorRts**

This member variable identifies the RTS that will monitor the RTS that receives this request.

FrGrReplayRequestProxy

class **FrGrReplayRequestProxy**

This class enables an external subsystem to send all types of Requests to RMS.

Public Functions

EcTVoid **GenReplayReq**(RWCString Originator, RWCString ScId, RWCString DbId,

RWCString RTSid, RWCString UserId, RWCString WksId, RWCString TlmType)

This member function will send a Replay Service Request to the RMS.

EcTVoid **GenStrDelete**(RWCString Originator, RWCString UserId, RWCString WksId, EcTInt StringId)

This member function will send a String Delete Request to the RMS.

EcTInt **Initialize**()

This member function will establish a connection with the RMS subsystem.

Private Data

Address* **myRmsAddress**

This member variable identifies the Address of the RMS String Manager process.

XDR **myXDR**

This member variable identifies the XDR stream to which a message/object will be passed

FrGrReplayServiceRequest

class **FrGrReplayServiceRequest**

This class is responsible for identify strings via a string table based on request attributes and StringID. This class also creates strings based on request attributes and creates Telemetry object, notifying the Telemetry object of their configuration. BEGIN_PROLOG

Base Classes

private **FrGrServiceRequest**

Public Construction

FrGrReplayServiceRequest(const FrGrReplayServiceRequest&)

This routine creates a duplicate of this class.

FrGrReplayServiceRequest()

This routine creates an instance of this class.

~FrGrReplayServiceRequest()

This routine deletes an instance of this class.

Public Functions

EcTInt **execute**(FrGrController* Controller)

This routine is responsible for containing all functionality necessary for processing the request. It calls additional subroutines in order to accomplish this task.

Protected Functions

EcTInt **createParamServer**()

This routine is responsible for creation of the Parameter Server object.

EcTInt **createReplayTlm**()

This routine is responsible for creation of the Telemetry Object(s) and notification of the Telemetry Object(s) that it needs to configure a particular telemetry process.

FrGrString* **makeString**()

This routine makes a particular string based on attributes of the request.

Private Data

RWCString **myDataType**

Identifies type of data

FrGrRequest

class **FrGrRequest**

This class is a generalization of RMS requests.

Public Construction

FrGrRequest(const FrGrRequest&)

FrGrRequest(const FrGrRequest&)

This member function creates a duplicate of this class.

FrGrRequest()

This member function is the default constructor for this class.

~FrGrRequest()

This member function is the destructor for this class.

Public Functions

virtual int **execute**(FrGrController* Controller)

This member function is a pure virtual operation that ensures that all children of this object will contain an execute operation.

Private Data

FrGrString* **myCurrentString**

This attribute points to the string associated with the request.

RWCString **myOriginator**

This attribute identifies the sender of the request.

FrGrRequestHandler

class **FrGrRequestHandler**

This class will receive a message/object on an XDR stream and reconstruct the appropriate object.

Public Construction

FrGrRequestHandler(const FrGrRequestHandler&)

FrGrRequestHandler(const FrGrRequestHandler&)

This member function creates a duplicate of this class.

FrGrRequestHandler()

This member function is the default constructor for this class.

~FrGrRequestHandler()

This member function is the destructor for this class.

Public Functions

FrGrRequest* **CheckQueue**()

This member function will return a Request object from the Queue.

EcTInt **Initialize**(FrGrController* Controller)

This member function will create a Queue for it's Requests. It will set it's address as well.

EcTInt **receiveRequest**(RWCollectable* newRequest)

This member function will receive a Request from FUI or ANALYSIS, instantiate a FrGrRequest object, and put that Request in a Queue.

EcTInt **sendStatus**(RWCollectable* sentRequest)

This member function will send a status for the passed Request.

Private Data

FdEvEventLogger* **myEventLogger**

This member variable points to the FdEvEventLogger.

RWSet* **myRequestQueue**

This member variable points to the Queue that contains string requests.

FrGrRmsFuiRequestProxy

class **FrGrRmsFuiRequestProxy**

This class enables the FUI subsystem to send all types of Requests to RMS.

Public Functions

EcTVoid **GenAdjustLimitReq**(RWCString UserId, RWCString WksId, EcTInt StringId,

EcTInt ParameterId, RWCString Type, EcTInt ParamValue, EcTInt TlmType, EcTInt SetId)

This member function will send an Adjust Limit Request to the RMS.

EcTVoid **GenBkupServReq**(EcTInt StringId, EcTInt RTSid, RWCString UserId, RWCString WksId)

This member function will send a Backup Service Request to the RMS.

EcTVoid **GenCommandPriviledgeReq**(RWCString UserId, RWCString WksId, EcTInt StringId)

This member function will send a Command Priviledge Request to the RMS.

EcTVoid **GenGroundControlPriviledgeReq**(RWCString UserId, RWCString WksId, EcTInt StringId)

This member function will send a Ground Control Priviledge Request to the RMS

EcTVoid **GenRTServReq**(RWCString ScId, RWCString DbId, RWCString RTSid, RWCString Mode, RWCString UserId, RWCString WksId)

This member function will send a Realtime Service Request to the RMS.

EcTVoid **GenSimulationServReq**(EcTInt Mode, RWCString Scid, RWCString Dbid, RWCString RTSid, RWCString UserId, RWCString WksId)

This member function will send a Simulation Service Request to the RMS.

EcTVoid **GenStrDelete**(RWCString UserId, RWCString WksId, EcTInt StringId)

This member function will send a String Delete Request to the RMS.

EcTVoid **GenStringConnectReq**(EcTInt StringId, RWCString UserId, RWCString WksId, RWCString TlmType, RWCString myUserType)

This member function will send a String Connect Request to the RMS.

EcTVoid **GenStringDisconnectReq**(EcTInt StringId, RWCString UserId, RWCString WksId)

This member function will send a String Disconnect Request to the RMS.

EcTVoid **GenStringFailOverReq**(RWCString UserId, RWCString WksId, EcTInt FailedStringId, RWCString FailedRTSid, RWCString BackupRTSid, EcTInt BackupStringId)

This member function will send a String Fail Over Request to the RMS.

EcTInt **Initialize**()

This member function will establish a connection with the RMS subsystem.

Private Data

Address* **myRmsAddress**

This member variable identifies the Address of the RMS String Manager process.

XDR **myXDR**

This member variable identifies the XDR stream to which a message/object will be passed

FrGrRmsWsRmsIF

class **FrGrRmsWsRmsIF**

This class is used to receive messages from the WS/RTS RMS subsystem.

Public Construction

FrGrRmsWsRmsIF(const FrGrRmsWsRmsIF&)

FrGrRmsWsRmsIF(const FrGrRmsWsRmsIF&)

This member function creates a duplicate of this class.

FrGrRmsWsRmsIF()

This member function is the default constructor for this class.

~FrGrRmsWsRmsIF()

This member function is the destructor for this class.

Public Functions

FrGrRequest* **CheckQueue**()

This member function will return a Request object from the Queue.

EcTInt **Initialize**(FrGrController* Controller)

This member function will create a Queue for it's Requests and possibly one for the string table updates. It will set it's address as well.

EcTInt **receiveRequest**(FrGrRequest* newRequest)

This member function will receive a Request from the WS/RTS RMS and put that Request in a Queue.

EcTInt **sendStatus**(FrGrRequest* sentRequest)

This member function will send a status for the passed Request.

Private Data

FdEvEventLogger* **myEventLogger**

This member variable points to the FdEvEventLogger.

RWCString **myHost**

This member variable identifies whether or not the RMS is on the WS

or on the RTS.

RWSet* **myRequestQueue**

This member variable points to the Queue that contains string requests.

Address* **myRmsAddress**

This member variable identifies the RMS Address of the RMS subsystem to which statuses will be sent.

FrGrRtsRmsRequestProxy

class **FrGrRtsRmsRequestProxy**

This class enables a RTS RMS to send Requests to the WS RMS.

Public Functions

EcTInt **Initialize**(FrGrController* Controller)

This member function will set a multicast address and create an XDR stream.

EcTInt **Multicast**(FrGrRequest* Request)

This member function will receive a status on the passed Request.

EcTInt **ReceiveStatus**(FrGrRequest* Request)

This member function will receive a status on the passed Request.

EcTInt **SendRequest**(FrGrRequest* Request, Address* WsRmsAddress)

This member function will send a Request to the WS RMS.

Private Data

Address* **myMulticastAddress**

This member variable identifies the Multicast Address.

XDR **myXDR**

This member variable identifies the XDR stream to which a message/object will be passed

FrGrServiceRequest

class **FrGrServiceRequest**

This class creates and deletes instances of this class

Base Classes

public **FrGrRequest**

Public Construction

FrGrServiceRequest(const FrGrServiceRequest&)

This routine creates a duplicate of this class.

FrGrServiceRequest()

This routine creates an instance of this class.

~FrGrServiceRequest()

This routine deletes an instance of this class.

Protected Functions

FrGrString* **makeString**()

makeString()

Private Data

RWCString **myDBid**

This member variable contains the database id

RWCString **myRTSid**

This member variable contains id of real-time server

RWCString **myScid**

This member variable contains the spacecraft id

RWCString **myUserid**

This member variable contains the id of the user

RWCString **myWksid**

This member variable contains the id of the Workstation

FrGrSimulationServiceRequest

class **FrGrSimulationServiceRequest**

_FrGrSimulationServiceRequest_h_

This class is responsible for processing a Simulation Service Request.

Base Classes

private **FrGrServiceRequest**

Public Construction

FrGrSimulationServiceRequest(const FrGrSimulationServiceRequest&)

This routine creates a duplicate of this class.

FrGrSimulationServiceRequest()

This routine creates an instance of this class.

~FrGrSimulationServiceRequest()

This routine deletes an instance of this class.

Public Functions

EcTInt **execute**(FrGrController* Controller)

This routine is responsible for containing all functionality necessary for processing the request. It calls additional subroutines in order to accomplish this task.

Protected Functions

EcTInt **createGSC**()

This routine is responsible for creation of the Ground Script Controller Object and notification of the Ground Script Controller Object that it needs to configure a particular Ground Script Controller process.

EcTInt **createParamServer**()

This routine is responsible for the creation of the Parameter Server Object.

FrGrString* **findString**(RWSet* PassedStringTable)

This routine finds a particular string in a string table based on the request attributes.

FrGrString* **makeString**()

This routine makes a particular string based on attributes of the request.

Private Data

RWCString **myMode**

This member variable is the identifier of the string's mode. This can be operational, test, or training.

FrGrSnapshotCompNotif

class **FrGrSnapshotCompNotif**

This class is responsible for containing all functionality necessary for processing the Snapshot Completion Notification.

Base Classes

public **FrGrRequest**

Public Construction

FrGrSnapshotCompNotif(const FrGrSnapshotCompNotif&)

This routine creates a duplicate of this class.

FrGrSnapshotCompNotif()

This routine creates an instance of this class.

~FrGrSnapshotCompNotif()

This routine deletes an instance of this class.

Public Functions

EcTInt **execute**(FrGrController* Controller)

This routine is responsible for containing all functionality necessary for processing the request. It calls additional subroutines in order to accomplish this task.

Protected Functions

FrGrString* **findString**(RWSet* PassedStringTable)

This routine finds a particular string in a string table based on the StringId.

Private Data

EcTInt **myStringID**

This member variable identifies the logical string for which this backup string is to be created.

RWCString **myTlmConfigFilename**

This member variable identifies the configuration filename for the Telemetry process.

FrGrSoftware

class **FrGrSoftware**

This class is a virtual base class that contains the ScId, DbId, RMS address and the State of the children.

Public Construction

FrGrSoftware(const FrGrSoftware&)

FrGrSoftware(const FrGrSoftware&)

This member function creates a duplicate of this class.

FrGrSoftware()

This member function is the default constructor for this class.

~FrGrSoftware()

This member function is the destructor for this class.

Protected Functions

EcTInt **Stop**()

This member function is pure virtual and makes this class abstract.

Private Data

RWCString **myDbId**

This member variable identifies the DbId that is used to configure the RMS configurable processes.

Address* **myRmsAddress**

This member variable identifies the address of the RMS process that the RMS configurable processes will communicate with.

RWCString **myScId**

This member variable identifies the ScId that is used to configure the RMS configurable processes.

Private Types

enum **myState**

This member variable identifies whether the RMS configurable processes are

in an active, inactive or a backup state. It is also used to configure the RCM processes.

Enumerators

ACTIVE
BACKUP
INACTIVE

FrGrStrManResMonProxy

class **FrGrStrManResMonProxy**

This class enables a String Manager to send Requests to the Resource Monitor.

Public Functions

EcTInt **Initialize**(FrGrController* Controller)

This member function will establish a connection with the Resource Monitor.

EcTInt **ReceiveStatus**(FrGrMonitorRequest* Request)

This member function will receive a status on the passed Request.

EcTInt **SendRequest**(FrGrMonitorRequest* Request)

This member function will send a Request to the RTS RMS.

Private Data

Address* **myResMonAddress**

This member variable identifies the Address of the Resource Monitor

process.

XDR **myXDR**

This member variable identifies the XDR stream to which a message/object will be passed

FrGrStringAccessRequest

class **FrGrStringAccessRequest**

This class finds a particular string in a string table based on the StringID.

Base Classes

public **FrGrRequest**

Public Construction

FrGrStringAccessRequest(const FrGrStringAccessRequest&)

FrGrStringAccessRequest(const FrGrStringAccessRequest&)

This routine creates a duplicate of this class.

FrGrStringAccessRequest()

This routine creates an instance of this class.

~FrGrStringAccessRequest()

This routine deletes an instance of this class.

Protected Functions

FrGrString* **indString**(RWSet* PassedStringTable, int PassedStringID)

findString

This routine finds a particular string in a string table based on the StringId.

Private Data

EcTInt **myStringId**

This member variable is the identifier of the string

RWCString **myUserId**

This member variable is the identifier of the user

EcTInt **myWksId**

This member variable is the identifier of the Workstation

FrGrStringConnectRequest

class **FrGrStringConnectRequest**

FrGrStringConnectRequest::FrGrStringConnectRequest()

This class is used to create a mirrored and tailored telemetry object, add a mirrored or tailored workstation, or add a user to the user list.

Base Classes

private **FrGrStringAccessRequest**

Public Construction

FrGrStringConnectRequest()

This routine creates an instance of this class.

~FrGrStringConnectRequest()

This routine deletes an instance of this class.

Public Functions

EcTInt **execute**(RWSet* passedStringTable, RWCString passedHostID, FrGrStrManResMonIF* passedStrManResMonIF, FrGrRmsWsRmsIF* passedFrGrRmsWsRmsIF, RWSet* passedFrGrRmsWsRmsIFset, FoGnRmsFuiIF* passedFoGnRmsFuiIF, FoGnRmsCsmsIF* passedFoGnRmsCsmsIF)

This routine is responsible for containing all functionality necessary for processing the request. It calls additional subroutines in order to accomplish this task.

Protected Functions

EcTInt **addMirroredWS**()

This routine adds a mirrored workstation to a string's mirrored WS list.

EcTInt **addTailoredWS**()

This routine adds a tailored workstation to a string's tailored WS list.

EcTInt **addUser**()

This routine adds a user to a string's user list.

EcTInt **createParamServer**()

This routine is responsible for creating a Parameter Server

EcTInt **createTelemetry**()

This routine is responsible for creating a telemetry object. In addition, it notifies the telemetry object to create a telemetry process.

FrGrTelemetry* **findTlm**(RWCString passedTlmType)

This routine finds a particular telemetry object based on the passedTlmType

Private Types

enum **myTlmType**

This member variable identifies the telemetry type

Enumerators

HEALTH
HOUSE_KEEPING

enum **myUserType**

This member variable identifies the type of user

Enumerators

MIRRORED
SHARED
TAILORED

FrGrStringDeleteRequest

class **FrGrStringDeleteRequest**

This class is responsible for processing a String Delete Request.

Base Classes

public **FrGrRequest**

Public Construction

FrGrStringDeleteRequest(const FrGrStringDeleteRequest&)

FrGrStringDeleteRequest(const FrGrStringDeleteRequest&)

This member function creates a duplicate of this class.

FrGrStringDeleteRequest()

This member function is the default constructor for this class.

~FrGrStringDeleteRequest()

This member function is the destructor for this class.

Public Functions

EcTInt **execute**(RWSet* passedStringTable, RWCString passedHostID, FrGrStrManResMonIF* passedStrManResMonIF, FrGrRmsWsRmsIF* passedFrGrRmsWsRmsIF, RWSet* passedFrGrRmsWsRmsIFset, FoGnRmsFuiIF* passedFoGnRmsFuiIF, FoGnRmsCsmsIF* passedFoGnRmsCsmsIF)

This routine is responsible for containing all functionality necessary for processing the request. It calls additional subroutines in order to accomplish this task.

Protected Functions

int **deleteCmd**(FrGrStrManResMonIF* StrManResMonIF)

This routine will delete the command object associated with a string. The software object destruction will also delete the command process it is associated with. If necessary, it will

notify the Resource Monitor Task to stop monitoring the process.

int **deleteGsc**(FrGrStrManResMonIF* StrManResMonIF)

This routine will delete the Ground Script Controller

object associated with a string.

The software object destruction will also delete the GSC process it is associated with. It will notify the Resource Monitor Task to stop monitoring the process.

int **deleteParamServer**(RWCString Host, FrGrStrManResMonIF* StrManResMonIF)

This routine will delete the Parameter Server

object associated with a string.

The software object destruction will also delete the PS process it is associated with. If necessary, it will notify the Resource Monitor Task to stop monitoring the process.

int **deleteRcm**(FrGrStrManResMonIF* StrManResMonIF)

This routine will delete the RTContact object associated with a string. The software object destruction will also delete the RTContact process it is associated with. If necessary, it will notify the Resource Monitor Task to stop monitoring the process.

int **deleteTlm**(RWCString Host, FrGrStrManResMonIF* StrManResMonIF)

This routine will delete any telemetry objects associated with a string. The software object destruction will also delete the telemetry processes they are associated with. If necessary, it will notify the Resource Monitor Task to stop monitoring the process.

FrGrString* **findString**(RWSet* PassedStringTable, int PassedStringID)

This routine finds a particular string in a string table based on the StringId.

int **removeAll**(FrGrStrManResMonIF* PassedStrManResMonIF, RWCString PassedHostId)

This routine will call the necessary subroutines for deleting software objects and processes.

int **removeString**(RWSet* PassedStringTable)

This routine will remove a string from the PassedStringTable.

Private Data

EcTInt **myStringId**

This attribute identifies the string to be deleted.

RWCString **myUserId**

This attribute identifies the user that generated the request.

RWCString **myWksId**

This attribute identifies the workstation that generated the request.

FrGrStringDisconnectRequest

class **FrGrStringDisconnectRequest**

FrGrStringDisconnectRequest::FrGrStringDisconnectRequest()

This class is used to delete a mirrored and tailored telemetry object, delete a mirrored or tailored workstation, or delete a user from the user list.

Base Classes

private **FrGrStringAccessRequest**

Public Construction

FrGrStringDisconnectRequest()

This routine creates an instance of this class.

~FrGrStringDisconnectRequest()

This routine deletes an instance of this class.

Public Functions

EcTInt **execute**(FrGrController* Controller)

This routine is responsible for containing all functionality necessary for processing the request. It calls additional subroutines in order to accomplish this task.

Protected Functions

EcTInt **removeAllTlm**(RWCString passedTlmType)

This routine removes all telemetry processes associated with a particular string off of a workstation.

EcTInt **removeParamServer**()

This routine removes the Parameter Server process associated with a particular string off of a workstation.

EcTInt **removeUser**()

This routine removes the UserId from a string's UserId list.

EcTInt **removeUserStation**()

This routine removes the WksId from a string's UserStation list.

FrGrStringFailoverRequest

class **FrGrStringFailoverRequest**

This class is responsible for processing a String Failover Request.

Base Classes

public **FrGrRequest**

Public Construction

FrGrStringFailoverRequest(const FrGrServiceRequest&)

This routine creates a duplicate of this class.

FrGrStringFailoverRequest()

This routine creates an instance of this class.

~FrGrStringFailoverRequest()

This routine deletes an instance of this class.

Public Functions

EcTInt **execute**(FrGrController* Controller)

This member function is responsible for processing this particular request.

Protected Functions

EcTInt **DeactivateTelemetry**()

This member function will initiate changing Telemetry's state from active to backup.

EcTInt **activateCommand**()

This member function will initiate changing Command's state from backup to active.

EcTInt **activateRTContact**()

This member function will initiate changing RTContact's state from backup to active.

EcTInt **activateTelemetry**()

This member function will initiate changing Telemetry's state from backup to active.

EcTInt **deactivateCommand**()

This member function will initiate changing Command's state from active to backup.

EcTInt **deactivateRTContact**()

This member function will initiate changing RTContact's state from active to backup.

FrGrString* **findString**(FrGrStringTable* PassedStringTable, int PassedStringID)

This member function will find a particular string within the string table based on its string ID.

FrGrTelemetry* **findTlm**(RWCString passedTlmType)

This member function will find a FrGrTelemetry object based on its type.

Private Data

RWCString **myActionFlag**

This member variable identifies whether the RTS needs to Activate or deactivate processes.

EcTInt **myBackupRTSid**

This member variable contains the RTSid to be failed over to.

EcTInt **myBackupStringId**

This member variable identifies the backup string to be failed over to.

EcTInt **myFailedRTSid**

This member variable contains the RTSid to be failed over from.

EcTInt **myFailedStringId**

This member variable identifies the failed string to be failed over from.

RWCString **myUserid**

This member variable contains the id of the user

RWCString **myWksid**

This member variable contains the id of the Workstation

FrGrStringStateUpdateRequest

class **FrGrStringStateUpdateRequest**

This class updates a string's state when its RTS fails.

Base Classes

public **FrGrRequest**

Public Construction

FrGrStringStateUpdateRequest(const FrGrStringStateUpdateRequest&)

FrGrStringStateUpdateRequest(const FrGrStringStateUpdateRequest&)

This routine creates a duplicate of this class.

FrGrStringStateUpdateRequest()

This routine creates an instance of this class.

~FrGrStringStateUpdateRequest()

This routine deletes an instance of this class.

Public Functions

EcTInt **execute**(FrGrController* Controller)

This routine is responsible for processing this request

FrGrString* **findString**(RWSet* PassedStringTable, int PassedStringID)

This routine will find a string from the PassedStringTable

Private Data

RWCString **myNewState**

This member variable identifies the new state of the string.

EcTInt **myStringId**

This member variable identifies the string that will be updated with the new state.

FrGrTableUpdateRequest

class **FrGrTableUpdateRequest**

This class updates the String Table with the String Table subset.

Base Classes

public **FrGrRequest**

Public Construction

FrGrTableUpdateRequest(const FrGrTableUpdateRequest&)

FrGrTableUpdateRequest(const FrGrTableUpdateRequest&)

This routine creates a duplicate of this class.

FrGrTableUpdateRequest()

This routine creates an instance of this class.

~FrGrTableUpdateRequest()

This routine deletes an instance of this class.

Public Functions

EcTInt **execute**(FrGrController* Controller)

This routine is responsible for processing this request

Private Data

RWSet **myTableSubset**

This member variable points to the string table subset that the WS String Table will be updated with.

FrGrTelemetry

class **FrGrTelemetry**

This class is used to create, destroy, configure and reconfigure the telemetry (TLM) subsystem. In addition, it will take a configuration snapshot of the TLM subsystem.

Base Classes

public **FrGrSoftware**

Public Construction

FrGrTelemetry(const FrGrTelemetry&)

This member function is a "copy constructor", it creates a duplicate of this class.

FrGrTelemetry(EcTInt TlmType)

FrGrTelemetry(RWCString TlmType)

This member function is the default constructor for this class.

FrGrTelemetry()

This member function is the default constructor for this class.

~FrGrTelemetry()

This member function is the destructor for this class. It will call the FrGrTelemetry::Stop() member function.

Public Functions

EcTInt **ChangeState**(State)

This member function will change the myState attribute of this class as well as send the change to the TLM processes.

EcTInt **Config**(RWCString DiagnosticFile, RWCString HKDecommFile, RWCString HSDecommFile, RWCString SBDecommFile)

Config(RWCString DiagnosticFile, RWCString HKDecommFile,

RWCString HSDecommFile, RWCString SBDecommFile)

This member function calls the MakeRmsAddress subroutine before calling the TLM StartTlmProcess() These operations will start and configure the corresponding TLM processes.

void **Config**(void)

This member function calls the MakeRmsAddress member function before calling TLM StartTlmProcess() These operations will start and configure the corresponding TLM processes.

EcTInt **Reconfig**(RWCString configParameters)

This member function will determine which TLM process will receive a configuration parameter and call the appropriate TLM ReconfigTlmProcess(RWCString configParam) member function

EcTInt **Snap**(void)

This member function will notify one of the TLM processes to take a configuration snapshot of their corresponding processes and update their mySnapFilename attribute.

EcTInt **Stop**(void)

This member function will call FrGrDiagnostic::StopDiagnostic()

Protected Functions

EcTInt **MakeRmsAddress**(void)

This member function will set the myRmsAddress attribute which

will be used to start the TLM processes.

Private Data

FrGrTelemetryProcess* **myDiagnostic**

This member variable points to the Diagnostic object.

Address* **myDmsAddress**

This member variable identifies the address of the DMS process that the TLM processes will communicate with.

FrGrTelemetryProcess* **myHKDecomm**

This member variable points to the HK Decomm object.

FrGrTelemetryProcess* **myHSDecomm**

This member variable points to the HS Decomm object.

FrGrTelemetryProcess* **mySBDecomm**

This member variable points to the SB Decomm object.

FrGrTelemetryProcess* **myStateCheck**

This member variable points to the State Check process object.

EcTInt **myTlmType**

This member variable contains the type of telemetry process is being executed.

FrGrTelemetryProcess

class **FrGrTelemetryProcess**

This class us used to create, destroy, configure and reconfigure any of the potential telemetry processes, i.e. Dump, StandBy, Housekeeping or Health & Safety. It will also take a configuration snapshot of any of the aforementioned processes.

Base Classes

public **FrGrTelemetry**

Public Construction

FrGrTelemetryProcess(const FrGrTelemetryProcess&)

This member function is a "copy constructor", it creates a duplicate of this class.

FrGrTelemetryProcess(RWCString myDbid, RWCString myScid, myState, Address* myRmsAddress)

This member function is the default constructor for this class.

~FrGrTelemetryProcess()

This member function is the destructor for this class. It will call the FrGrTelemetryProcess::StopTlmProcess() member function.

Public Functions

EcTInt **ConfigTlmProcess**()

This member function sends configuration information to the FrGrTelemetryProcess process after it has been started.

EcTInt **ReconfigTlmProcess**(RWCString configParameter)

This member function sends a reconfiguration parameter to the specified telemetry process.

EcTInt **SnapConfigTlmProcess**(RWCString ConfigFile)

This member function sends snapshot configuration information to a telemetry process after it has been started.

EcTInt **SnapTlmProcess**(void)

This member function requests a configuration snapshot of the FrGrTelemetyProcess process and updates the mySnapFilename attribute.

EcTInt **StartTlmProcess**()

This member function will create a telemetry process of one of the following telemetry types: Dump, StandBy, Housekeeping or Health & Safety.

Protected Functions

EcTInt **StopTlmProcess**(void)

This member function will terminate a FrGrTelemetryProcess process.

Private Data

FrGrDataArchiver* **myDataArchiver**

This member variable points to the FrGrDataArchiver object

RWCString **myDbId**

This member variable is the database identifier which is to be used by this process for the extraction of database information.

FtTlDumpConfig* **myDumpConfig**

This member variable points to the Telemetry Dump Proxy.

FrGrTelemetry* **myFrGrTelemetry**

This member variable points to the FrGrTelemetry object that this object is part of.

RWCString **myProcessType**

This member variable identifies whether the Telemetry Process is to process Housekeeping, Health&Safety, Standby, Diagnostic, or State Checking.

Address* **myRmsAddress**

This member variable points to the address of the RMS address.

RWCString **myScId**

This member variable is the identifier of the space craft of which this process is supporting.

RWCString **mySnapFilename**

This member variable identifies a snapshot configuration filename.

FtTlTelemetryConfig* **myTelemetryConfig**

This member variable points to the Telemetry proxy.

EcTInt **myTlmPid**

This member variable identifies the telemetry processes, Pid

FrGrWsRmsRequestProxy

class **FrGrWsRmsRequestProxy**

This class enables a WS RMS to send Requests to the RTS RMS.

Public Functions

EcTInt **Initialize**(FrGrController* Controller, RWCString RtsId)

This member function will establish a connection with the RTS RMS subsystem. It will create a Request Queue as well.

EcTInt **ReceiveStatus**(FrGrRequest* Request)

This member function will receive a status on the passed Request.

EcTInt **SendRequest**(FrGrRequest* Request)

This member function will send a Request to the RTS RMS.

Private Data

Address* **myRtsRmsAddress**

This member variable identifies the Address of the RTS RMS String Manager process.

XDR **myXDR**

This member variable identifies the XDR stream to which a message/object will be passed

3.3 RMS Resource Monitor Component

The Resource Monitor Component provides the Resource Management Subsystem's monitoring service for critical EOC resources. The Resource Management Subsystem Resource Monitor Component is designed to be installed and executed only on the Real-Time Servers within the

EOC. The role of the Resource Monitor is much the same from server to server and is dependent upon requests from the String Manager component that is co-located.

3.3.1 RMS Resource Monitor Component Context

The RMS Resource Monitor Component receives and responds to requests for hardware and software component monitoring from the RMS String Manager Component. The requests are for starting, stopping and modifying the monitoring from one component to another as the mission critical components change with the operating environment. As these operating conditions change (e.g. logical strings are created and deleted) and are reported by the String Manager, the changes in monitoring needs are relayed to the CSMS/MSS monitoring service by the Resource Monitor. The Resource Monitor registers interests in newly created software components and terminates interests in deleted software components. Hardware components are monitored automatically by CSMS/MSS.

Receiving notice of changed hardware and software component status's is more complex. When the status of a mission critical hardware or software component changes, a management event is generated by the CSMS/MSS Monitoring Service. The event is reported to the Data Management Subsystem where it is determined that the incoming event is of a type that the RMS Resource Monitor is interested. The DMS uses a Paramater Server interface class to translate the event information into a component status and report that status to the Parmeter Server. Interest in certain hardware and software status parameters is registered by the Resource Monitor process. Again, this depends on which hardware and software components are considered mission critical at a given time. When a failure status is received by the Resource Monitor process an event indicating a mission critical component failure will be generated.

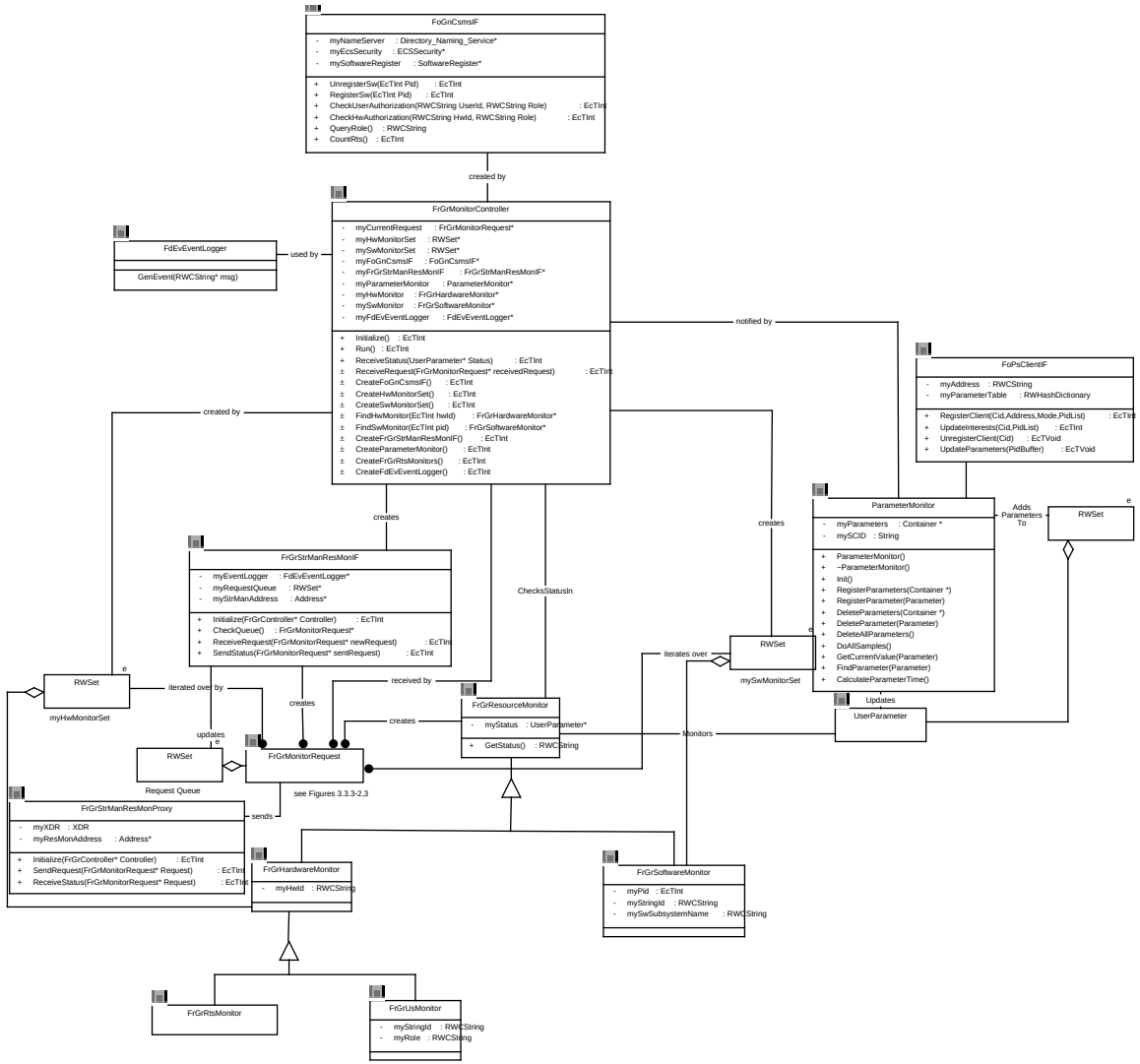


Figure 3.3.1-1. RMS Resource Monitor Component Context Diagram

3.3.2 RMS Resource Monitor Component Interfaces

Table 3.3.2 RMS Resource Monitor Component Interfaces

Interface Service	Interface Class	Interface Class Description	Service Provider	Service User	Frequency
RTS RMS to Resource Monitor Interface	FrGrStr ManRes MonProxy	Enables the RTS RMS to send requests to Resource Monitor	RMS Resource Monitor	RTS RMS	30 - 100 per contact
Un/Registering of SW and querying the name server	FoGnCsms IF	Enables RMS to interface with the name server	CSMS	RMS	24 - 80 per contact
Gets and Sets parameters from/ to PS	FoPsClient IF	Enables RMS to interface with PS	PS	RMS	30 - 100 per contact
Reads requests sent by the RMS String Mgr.	FrGrStr ManRes MonIF	Enables the RMS Resource Monitor to read requests from the String Mgr.	RMS	RMS Resource Monitor	30 - 100 per contact
Sends event messages to event logger	FdEvEventL ogger	Enables RMS to log event messages	DMS	RMS	1 per SW or HW failure

3.3.3 RMS Resource Monitor Component Object Model

Figure 3.3.3-1 illustrates a top level view of the RMS Resource Monitor Component. Subsequent Figures illustrate, in more detail, the FrGrMonitorRequest object. The objects shown on Figure 3.3.3-1 allow the RMS to collect status on software and hardware, register software with the CSMS name server, and generate events when a failure status is received.

The RWSet object is a Rogue Wave Collection Class that is used to store other RMS Resource Monitor objects. This includes Hardware and Software Monitor objects, User Parameter objects, and Monitor Request objects.

There are several "proxy" and "receiver" objects that appear on the diagram. The FrGrStrManResMonProxy object appears on the diagram for clarity. It resides in the RMS String Manager process and enables the String Manager to send MonitorRequests to Resource Monitor. The MonitorRequests are received by the FrGrStrManResMonIF object and placed into a queue. The queue is represented by a RWSet. When the Controller object invokes CheckQueue within the FrGrStrManResMonProxy, it retrieves a MonitorRequest object from the queue and returns it to the Controller. The FrGrStrManResMonIF will send a request status back to the String Manager when notified by the Controller. The FdEvEventLogger enables the Resource Monitor to send

events to DMS. The FoGnCsmsIF object enables the Resource Monitor to register and unregister software processes with the CSMS name server. The FoPsClientIF allows the Resource Monitor to receive status information on hardware and software from the Parameter Server.

There are three types of Monitors within the Resource Monitor. The FrGrRtsMonitor is derived from the FrGrHardwareMonitor and stores the status of the RTS. The FrGrUsMonitor is derived from the FrGrHardwareMonitor and stores the status of the userstations. In addition, it tracks the string associated with a userstation and what type of role the userstation plays within the EOC. The role could be Ground Control or Command Authority. The FrGrSoftwareMonitor is derived from the FrGrResourceMonitor and stores a Process ID, the ID of the String that the process belongs to, and a Software Subsystem Name. The Software Subsystem Name could be Telemetry, Command, RTContact, Ground Script Controller, Parameter Server, or Data Archiver. The FrGrHardwareMonitor object is derived from the FrGrResourceMonitor object and merely serves as a generalization of the FrGrRtsMonitor and the FrGrUsMonitor. It is an abstract class that only contains a HwId as an attribute. The FrGrResourceMonitor is an abstract class that serves as a generalization of all of the Monitor objects needed by the Resource Monitor.

The FrGrMonitorRequest object is an abstract class that contains a virtual Execute operation. Several Request objects are derived from the FrGrMonitorRequest object and each will overwrite the Execute operation. The Execute operation is called by the FrGrMonitorController object and is responsible for containing all functionality or calling any subroutines necessary for processing a particular request. The classes derived from the FrGrMonitorRequest object are shown in more detail in Figure 3.3.3-2 and Figure 3.3.3-3.

The ParameterMonitor object receives User Parameters through the FoPsClientIF and notifies the FrGrMonitorController of a change in the Parameter's value.

The FrGrMonitorController object enables the Resource Monitor to initialize itself, communicate with other processes, and initiate the processing of requests. At initialization, the FrGrMonitorController creates the appropriate "proxy" and "receiver" objects. In addition, it creates collection classes and FrGrRtsMonitor objects for every RTS, except for the RTS that this particular Resource Monitor resides on. Once the Controller has initialized, it enters a "run" state. In this state, the Controller will notify requests to execute as they are received from the String Manager. When a failure status is received from the Parameter Server, an event is sent to DMS.

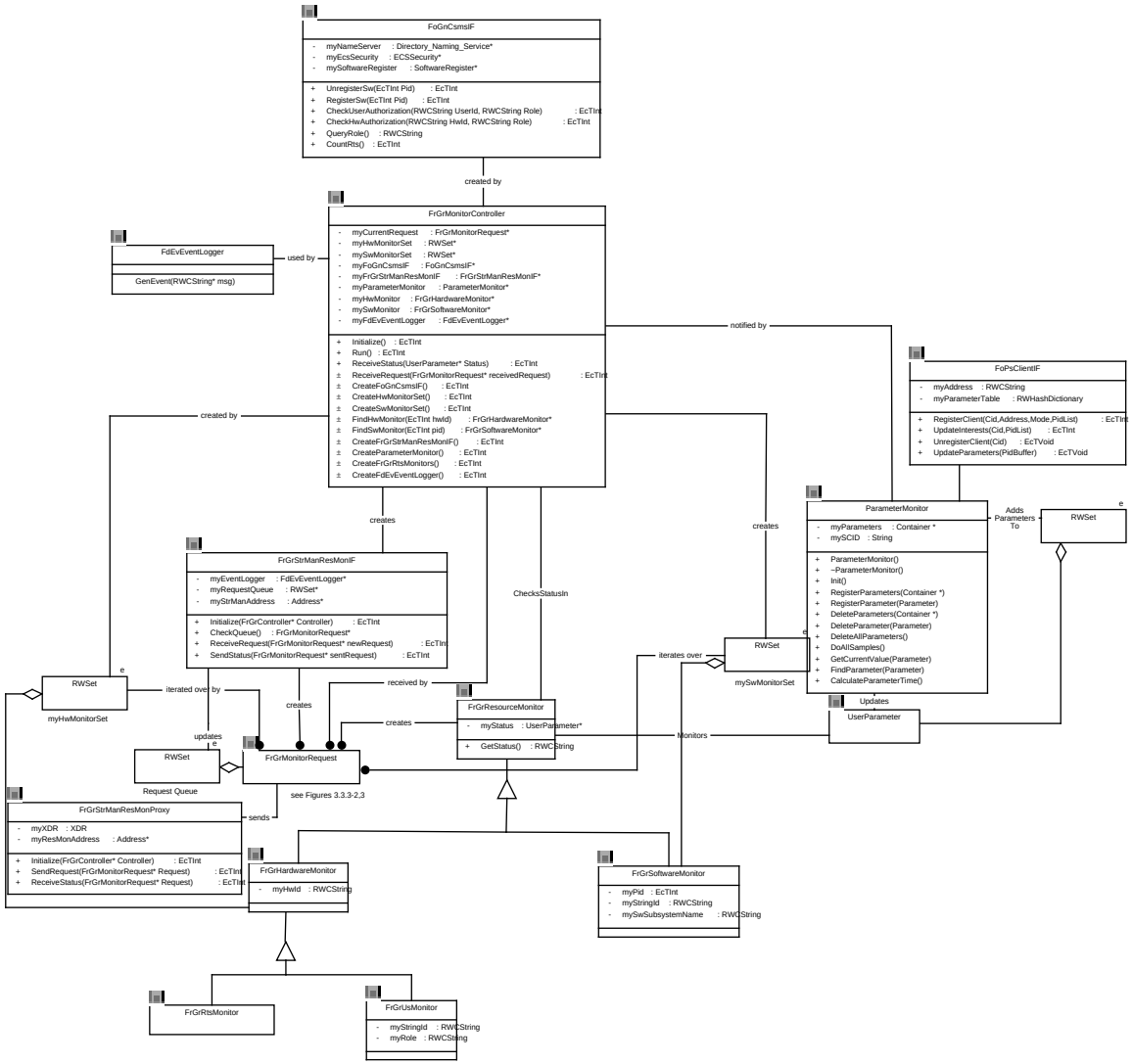


Figure 3.3.3-1. RMS Resource Monitor Component Object Model

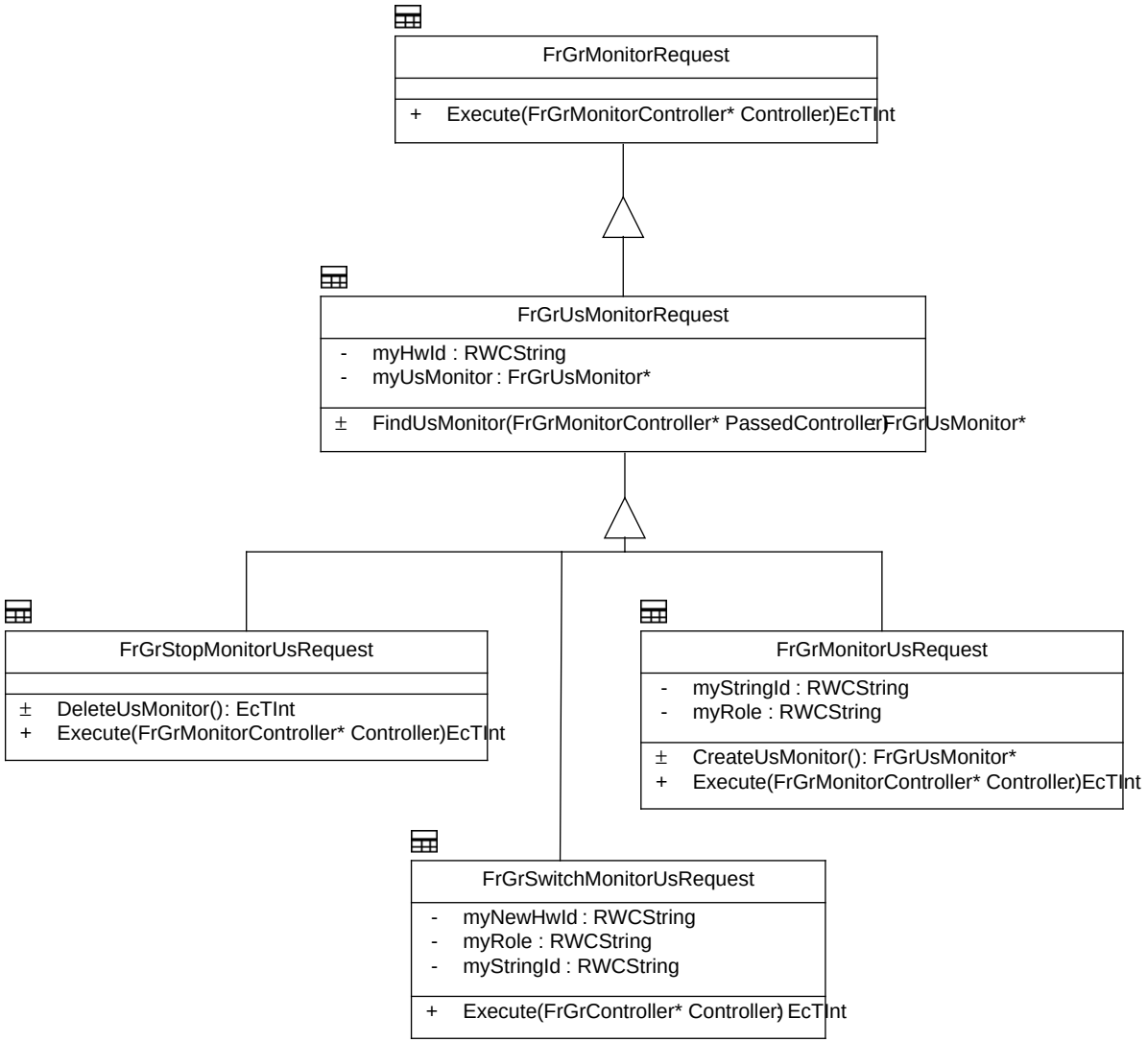


Figure 3.3.3-2. RMS Resource Monitor Component FrGrUsMonitorRequest Object Model

The FrGrUsMonitorRequest object is derived from the FrGrMonitorRequest object. FrGrMonitorRequest is an abstract class that contains an Execute operation that derived classes will inherit. FrGrUsMonitorRequest contains additional attributes and a FindUsMonitor operation that derived objects will inherit. The FrGrStopMonitorUsRequest object contains attributes and operations necessary for terminating the monitoring of a userstation. It will delete the FrGrUsMonitor object as well as notify the Parameter Server that there is no longer an interest in the userstation status. This would occur when a string is deleted from the RTS. The FrGrMonitorUsRequest object contains attributes and operations necessary for initiating the monitoring of a userstation. It will create the FrGrUsMonitor object as well as notify the Parameter Server that there an interest in the userstation status. This would occur when a string is created on the RTS. The FrGrSwitchMonitorUsRequest contains attributes and operations necessary to terminate the monitoring of one userstation and initiate the monitoring of another. It uses a FrGrUsMonitor object that has been already created, notifies the Parameter Server that there is no longer an interest in a particular userstation, and notifies it of a new interest in a different userstation. This would occur when the Ground Control or Command Authority user changes.

The FrGrSwMonitorRequest object is derived from the FrGrMonitorRequest object. FrGrMonitorRequest is an abstract class that contains an Execute operation that derived classes will inherit. FrGrSwMonitorRequest contains additional attributes that derived objects will inherit. The FrGrStopMonitorSwRequest object contains operations necessary for terminating the monitoring of a software process. It will delete the FrGrSoftwareMonitor object, notify the Parameter Server that there is no longer an interest in the software status, and notify CSMS to discontinue monitoring of the software process. This can occur when a string is deleted from the RTS. The FrGrMonitorSwRequest object contains attributes and operations necessary for initiating the monitoring of a software process. It will create the FrGrSoftwareMonitor object, notify the Parameter Server that there is an interest in the software status, and notify CSMS to begin monitoring the software process. This can occur when a string is created.

3.3.4 RMS Resource Monitor Component Dynamic Model

The following are the RMS Resource Monitor Component scenarios which are defined in this section.

- Request for Software Monitoring is Received by the Resource Monitor
- Request for User Station Monitoring is Received by the Resource Monitor
- Request to Switch User Station Monitoring is Received by the Resource Monitor
- Failed Hardware Status from the Parameter Server is Received by the Resource Monitor

3.3.4.1 Request for Software Monitoring is Received by the Resource Monitor Scenario

3.3.4.1.1 Request for Software Monitoring is Received by the Resource Monitor Abstract

The purpose of the Request for Software Monitoring is Received by the Resource Monitor scenario is to describe how the RMS Resource Monitor component acts upon a request for a software process to be monitored by the SCDO MSS Monitoring Service.

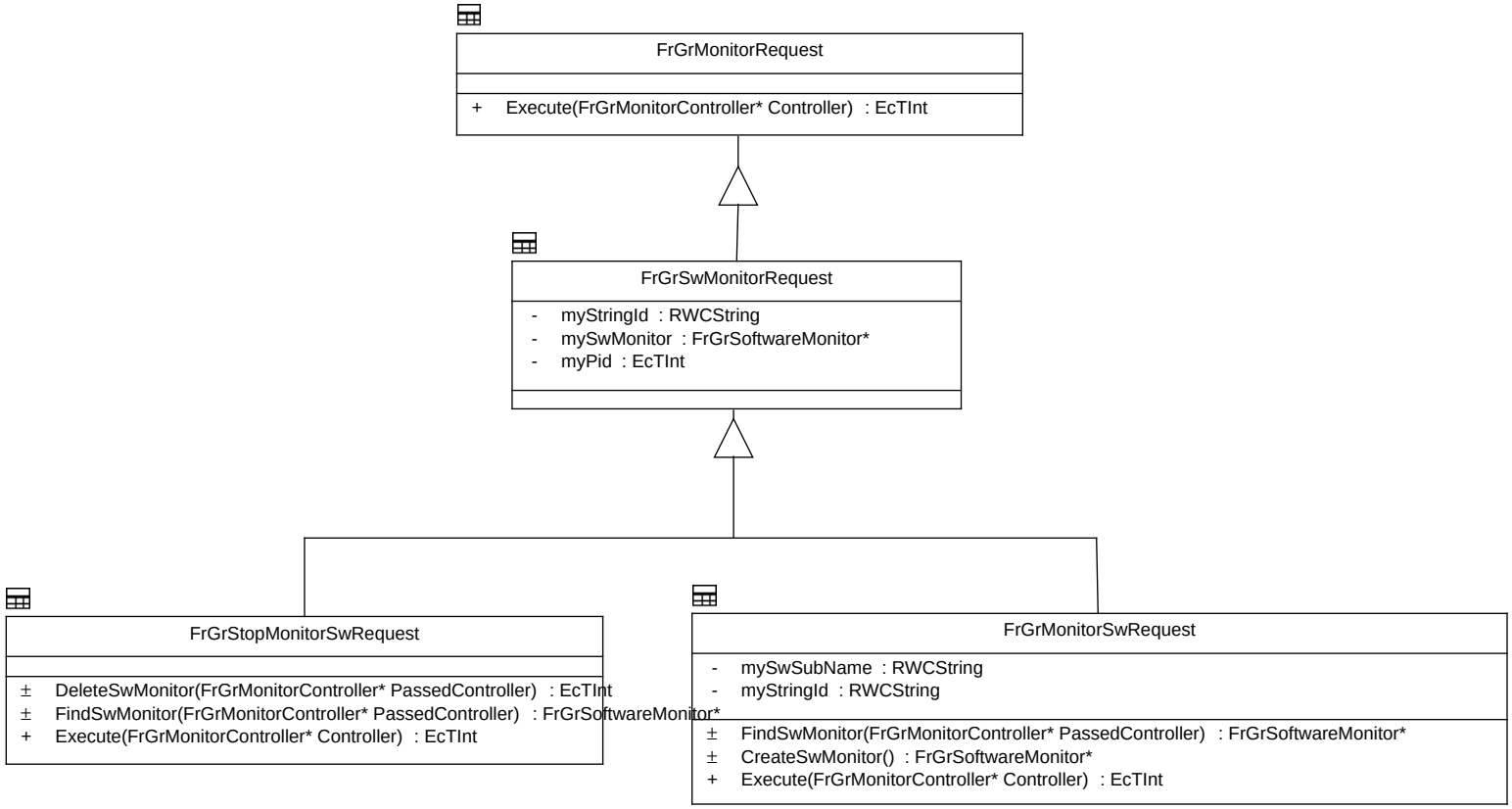


Figure 3.3.3-3. RMS Resource Monitor Component FrGrSwMonitorRequest Object Model

3.3.4.1.2 Request for Software Monitoring is Received by the Resource Monitor Summary Information

Interfaces:

SCDO/MSS Monitoring Service

Parameter Server

Stimulus:

The String Manager sends a FrGrMonitorSwRequest object to the Resource Monitor for processing.

Desired Response:

CSMS will be notified of a new software process to monitor and the Parameter Server will be notified to begin sending the Resource Monitor a status on the software process.

Pre-Conditions:

A software process has been created by the String Manager.

Post-Conditions:

The Resource Monitor is capable of receiving a software status from the Parameter Server and send an event if a software failure occurs.

3.3.4.1.3 Scenario Description

The Controller checks the queue of the FrGrStrManResMonIF and a FrGrMonitorSwRequest is returned. The MonitorController invokes the FrGrMonitorSwRequest object's Execute operation. A SwMonitor search is performed and a FrGrSoftwareMonitor object is not found. A FrGrSoftwareMonitor object is created and added to the SwMonitorSet. MSS is notified of the new software process and the Parameter Server is notified to send Resource Monitor the status of the new process. The Request is deleted and the String Manager is notified that the Request has been executed.

3.3.4.1.4 State Transition Description

3.3.4.2 Request for User Station Monitoring is Received by the Resource Monitor Scenario

3.3.4.2.1 Request for User Station Monitoring is Received by the Resource Monitor Abstract

The purpose of the Request for User Station Monitoring is Received by the Resource Monitor scenario is to describe how the RMS Resource Monitor component acts upon a request for a particular User Workstation to be monitored by the SCDO MSS Monitoring Service.

3.3.4.2.2 Request for User Station Monitoring is Received by the Resource Monitor Summary Information

Interfaces:

Parameter Server

Stimulus:

The String Manager grants Command Authority and sends a FrGrMonitorUsRequest to the Resource Monitor for processing.

Desired Response:

The Parameter Server will be notified to begin sending the Resource Monitor a status on the user station.

Pre-Conditions:

Command Authority has been granted on a particular userstation

Post-Conditions:

The Resource Monitor is capable of receiving a hardware status from the Parameter Server and send an event if a userstation failure occurs.

3.3.4.2.3 Scenario Description

The Controller checks the queue of the FrGrStrManResMonIF and a FrGrMonitorUsRequest is returned. The MonitorController invokes the FrGrMonitorUsRequest object's Execute operation. A UsMonitor search is performed and a FrGrUsMonitor object is not found. A FrGrUsMonitor object is created and added to the HwMonitorSet. The Parameter Server is notified to send Resource Monitor the status of the user station. The Request is deleted and the String Manager is notified that the Request has been executed.

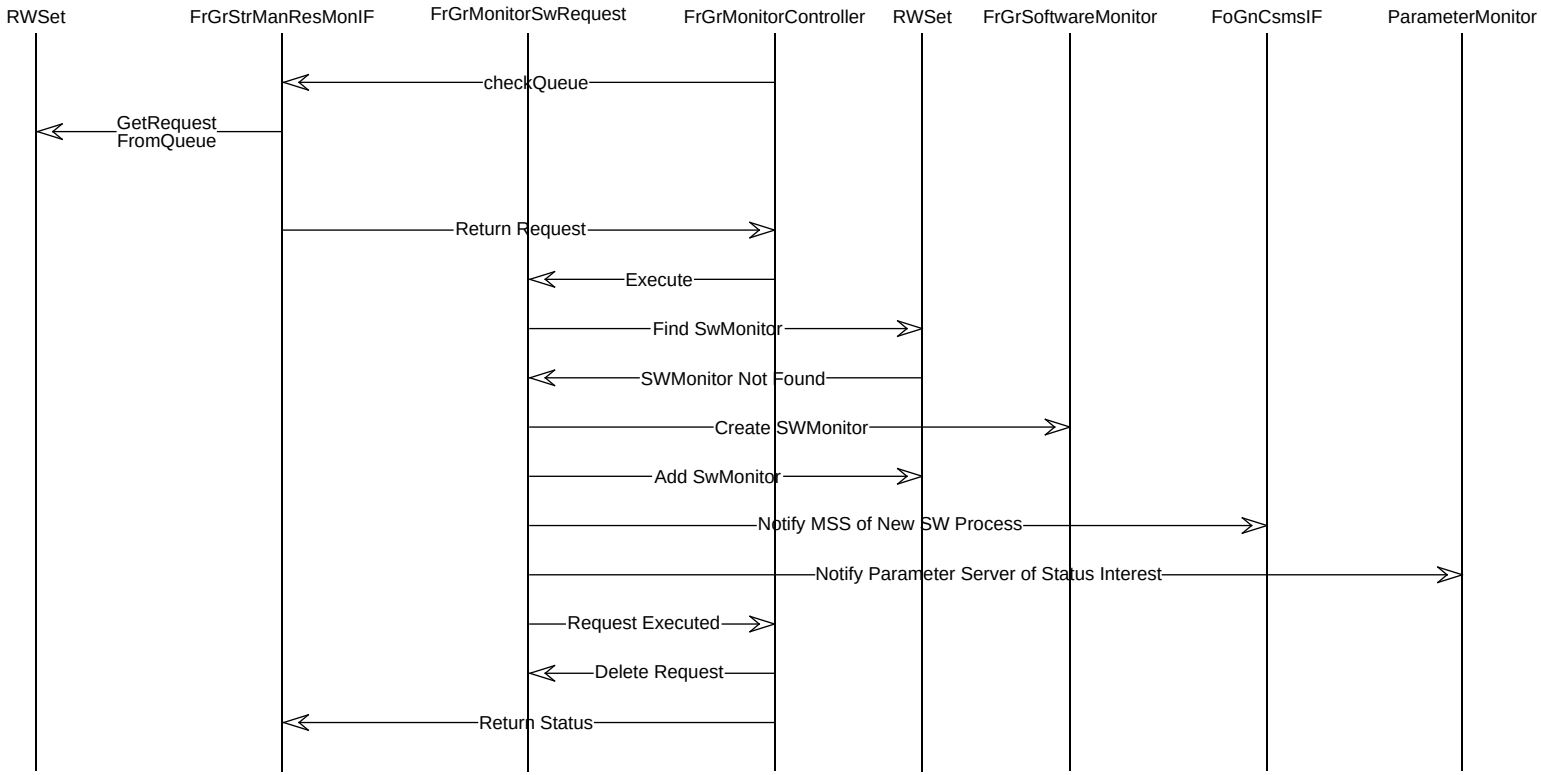


Figure 3.3.4.1.4-1. Request for Software Monitoring is Received by the Resource Monitor Event Trace

3.3.4.2.4 State Transition Description

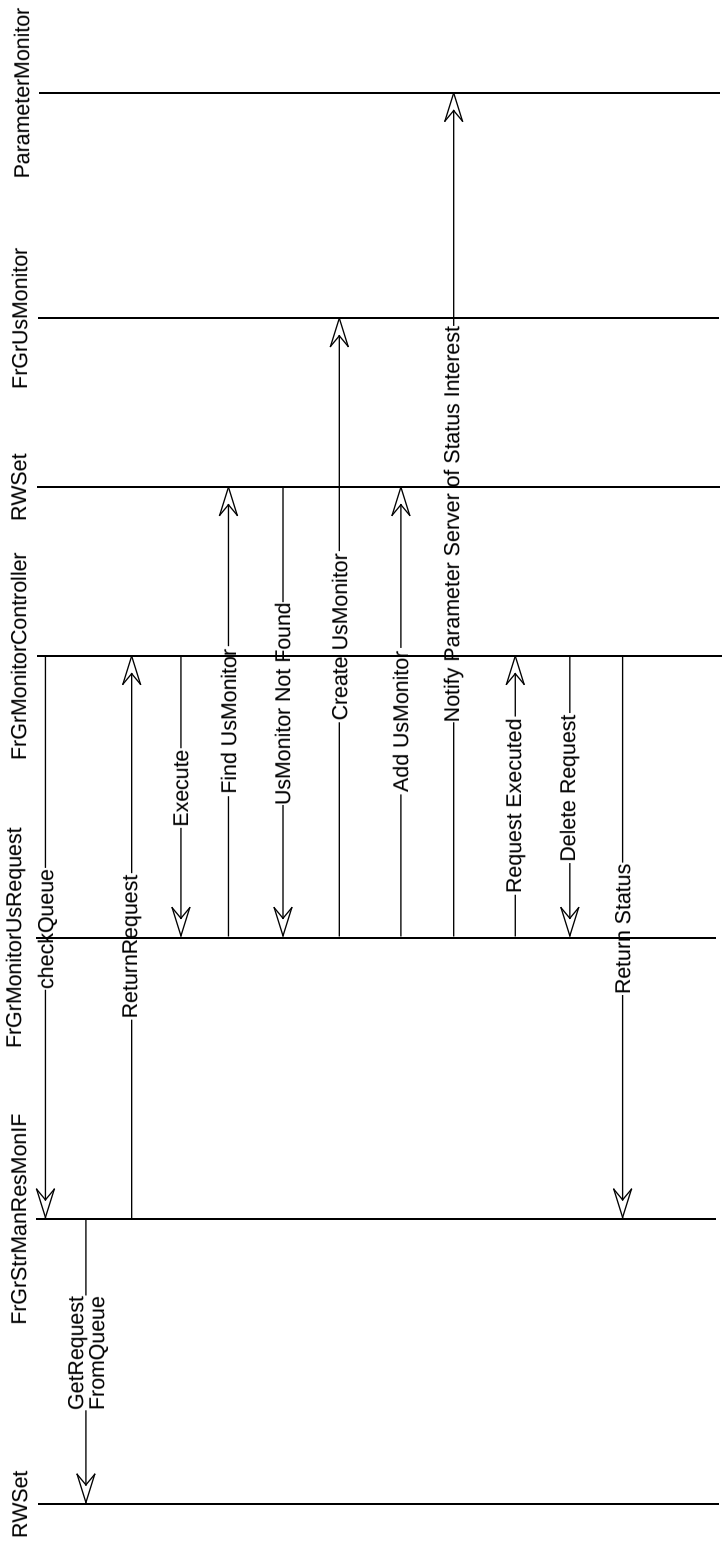


Figure 3.3.4.2.4-1. Request for User Station Monitoring is Received by the Resource Monitor Event Trace

3.3.4.3 Request to Switch User Station Monitoring is Received by the Resource Monitor Scenario

3.3.4.3.1 Request to Switch User Station Monitoring is Received by the Resource Monitor Abstract

The purpose of the Request to Switch User Station Monitoring is Received by the Resource Monitor scenario is to describe how the RMS Resource Monitor component acts upon receipt of a request to change the User Station monitoring that is provided by the SCDO MSS Monitoring Service. This essentially results in the RMS Resource Monitor terminating its interest in the status of one User Station in favor of interest in another User Station.

3.3.4.3.2 Request to Switch User Station Monitoring is Received by the Resource Monitor Summary Information

Interfaces:

Parameter Server

Stimulus:

The String Manager changes Command Authority and sends a FrGrSwitchMonitorUsRequest to the Resource Monitor for processing.

Desired Response:

The Parameter Server will be notified to discontinue sending the status on one userstation and begin sending the Resource Monitor a status on a different user station.

Pre-Conditions:

A different user has been granted Command Authority on a particular string and a different userstation will need to be monitored.

Post-Conditions:

The Resource Monitor is capable of receiving a hardware status from the Parameter Server on a different userstation and send an event if a userstation failure occurs.

3.3.4.3.3 Scenario Description

The Controller checks the queue of the FrGrStrManResMonIF and a FrGrSwitchMonitorUsRequest is returned. The MonitorController invokes the FrGrSwitchMonitorUsRequest object's Execute operation. A UsMonitor search is performed and a FrGrUsMonitor object is found. The Parameter Server is notified to stop sending Resource Monitor the status of the old user station. The FrGrUsMonitor's HwId is changed and the Parameter Server is notified to start sending the Resource Monitor the new user station status. The Request is deleted and the String Manager is notified that the Request has been executed.

3.3.4.3.4 State Transition Description

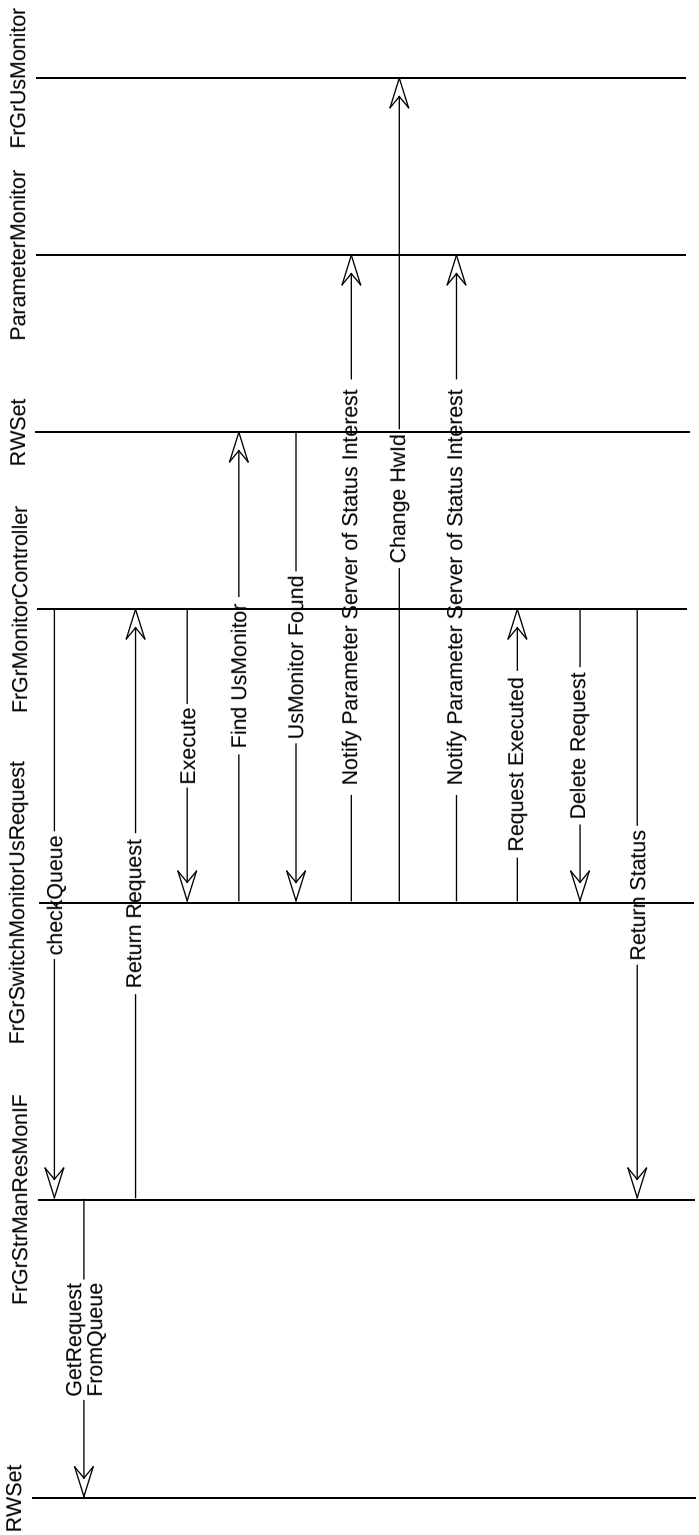


Figure 3.3.4.3.4-1. Request to Switch User Station Monitoring is Received by the Resource Monitor Event Trace

3.3.4.4 Failed Hardware Status from the Parameter Server is Received by the Resource Monitor Scenario

3.3.4.4.1 Failed Hardware Status from the Parameter Server is Received by the Resource Monitor Abstract

The purpose of the Failed Hardware Status from the Parameter Server is Received by the Resource Monitor scenario is to describe how the RMS Resource Monitor component acts upon notification from the Parameter Server that a hardware component has failed. The Parameter Server receives information about the registered components status Svia a proxy that is provided to the DMS Event Handler. The DMS Event Handler receives management events including changes in hardware component status from the SCDO MSS Monitoring Service.

3.3.4.4.2 Failed Hardware Status from the Parameter Server is Received by the Resource Monitor Summary Information

Interfaces:

Data Management Subsystem

Parameter Server

Stimulus:

The RMS Resource Monitor receives a FAILED HW Status from the Parameter Server.

Desired Response:

The RMS software will receive a HW failure status from the Parameter Server and send an event that a RTS has failed.

Pre-Conditions:

CSMS detects a RTS failure and notifies DMS. DMS will notify the Parameter Server of the status.

Post-Conditions:

Users will be notified that a RTS failure has occurred and appropriate failover actions can be taken.

3.3.4.4.3 Scenario Description

The ParameterMonitor object receives a HW Status from the Parameter Server. The MonitorController is notified of the status and determines that the status indicates a HW failure. The HwMonitor object is retrieved and it is determined that the HwMonitor object is monitoring a RTS. The FrGrMonitorController will then send a RTS failure event.

3.3.4.4.4 State Transition Description

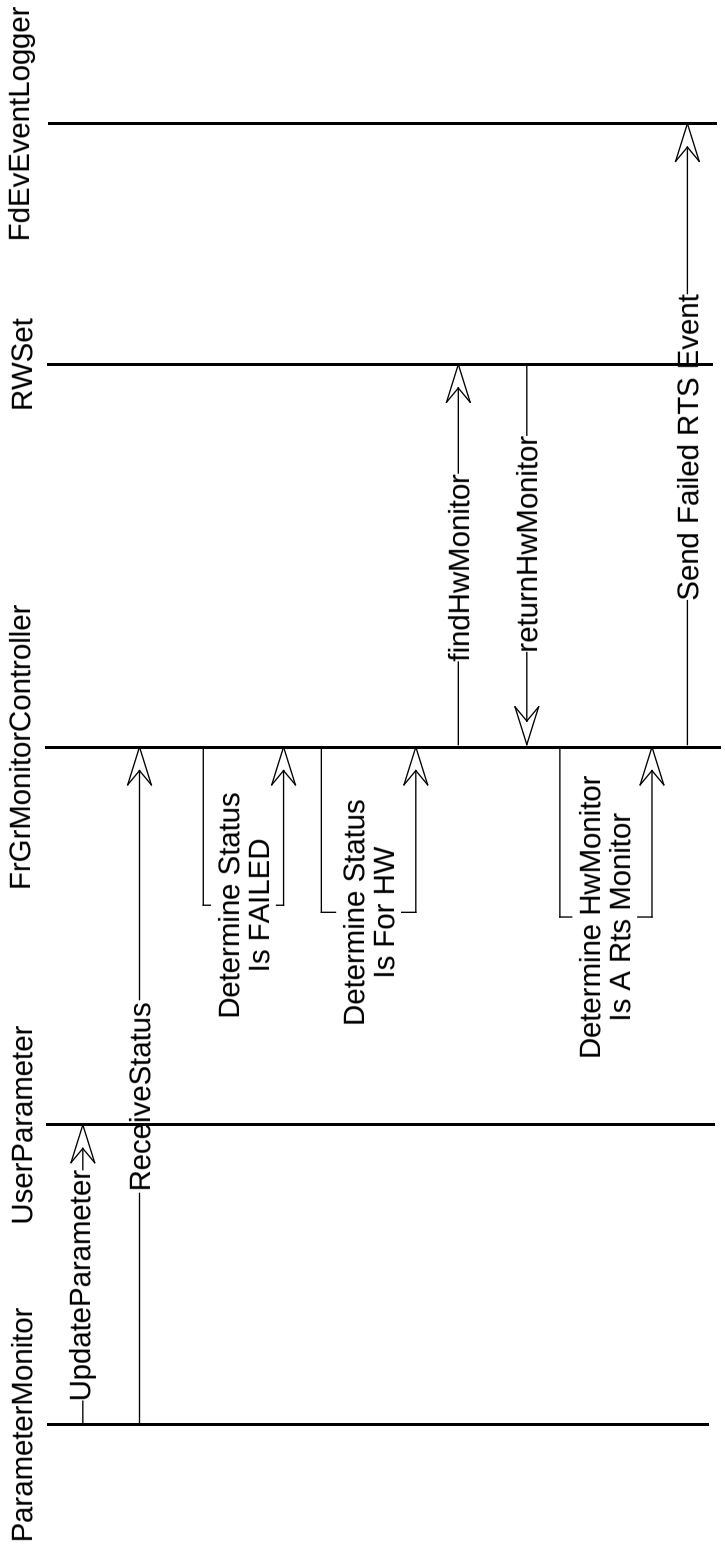


Figure 3.3.4.4.4-1. Failed Hardware Status from the Parameter Server is Received by the Resource Monitor Event Trace

3.3.5 RMS Resource Monitor Component Data Dictionary

FrGrHardwareMonitor

class **FrGrHardwareMonitor**

This is an abstract base class that is a generalization of the Software and Hardware Monitor objects.

Base Classes

public **FrGrResourceMonitor**

Private Data

RWCString **myHwId**

This attribute identifies the HwId of a particular piece of hardware.

FrGrMonitorController

class **FrGrMonitorController**

This class is responsible for initializing start-up of the Resource Monitor as well as initiating the processing of FrGrStatus and FrGrMonitorRequest objects.

Public Functions

EcTInt **Initialize**(void)

This member function is responsible for establishing interfaces with external subsystems as well as creating the necessary Collection Objects.

EcTInt **ReceiveRequest**(FrGrMonitorRequest* receivedRequest)

When a request is received by the FrGrStrManResMonIF object, this operation is invoked from the interface object. It is passed a pointer to the Request object.EcTInt **ReceiveStatus**(UserParameter* Status)

When a change in status is received by the ParameterMonitor object, this operation is invoked. It is passed the UserParameter object.

EcTInt **Run**(void)

This member function is responsible for checking the appropriate interfaces for passed data.

Protected Functions

EcTInt **CreateFdEvEventLogger**(void)

This operation creates the FdEvEventLogger object that will be passed to

the Event Handler whenever a need arises.

EcTInt **CreateFoGnCsmsIF**(void)

This operation creates the FoGnCsmsIF object and establishes a connection with CSMS.

EcTInt **CreateFrGrStrManResMonIF**()

This operation creates the FrGrStrManResMonIF object and establishes a connection with the String Manager process.

EcTInt **CreateHwMonitorSet**(void)

This operation creates the collection object that will be used to store the Hardware Monitor Objects.

EcTInt **CreateParameterMonitor**(void)

This operation creates the ParameterMonitor object and establishes a connection with the Parameter Server process.

EcTInt **CreateSwMonitorSet**(void)

This operation creates the collection object that will be used to store theSoftware Monitor Objects.

EcTInt **FrGrMonitorController::CreateFrGrRtsMonitors**()

CreateFrGrRtsMonitors

This operation will create a FrGrRtsMonitor object for every RTS except for this RTS at initialization

FrGrHardwareMonitor* **FrGrMonitorController::FindHwMonitor**(EcTInt hwId)

FindHwMonitor

This operation finds a particular FrGrHardwareMonitor object based on the passed HwId.

FrGrSoftwareMonitor* **FrGrMonitorController::FindSwMonitor**(EcTInt pid)FindSwMonitor

This operation finds a particular FrGrSoftwareMonitor object based on the passed PID.

Private Data

FrGrMonitorRequest* **myCurrentRequest**

This attribute points to a FrGrMonitorRequest object

FdEvEventLogger* **myFdEvEventLogger**

This attribute points to the FdEvEventLogger object.

FoGnCsmsIF* **myFoGnCsmsIF**

This attribute points to the FoGnCsmsIF object.

FrGrStrManResMonIF* **myFrGrStrManResMonIF**

This attribute points to the FrGrStrManResMonIF object. This object will facilitate an interface with the RTS String Manager process.

FrGrHardwareMonitor* **myHwMonitor**

This attribute points to a FrGrHardwareMonitor object

RWSet* **myHwMonitorSet**

This attribute identifies the RWSet that contains the Hardware Monitor objects.

ParameterMonitor* **myParameterMonitor**

This attribute points to the ParameterMonitor object.

FrGrSoftwareMonitor* **mySwMonitor**

This attribute points to a FrGrSoftwareMonitor object

RWSet* **mySwMonitorSet**

This attribute identifies the RWSet that contains the Software Monitor objects.

FrGrMonitorRequest

class **FrGrMonitorRequest**

This base class is a generalization of all of the Monitor Requests received from the String Manager process.

Public Functions

EcTInt **Execute**(FrGrMonitorController* Controller)

This is a virtual operation that ensures every derived class will define its own Execute operation.

FrGrMonitorSwRequest

class **FrGrMonitorSwRequest**

This class represents a request sent from the String Manager process to the Resource Monitor process to begin monitoring a created software process.

Base Classes

public **FrGrSwMonitorRequest**

Public Functions

EcTInt **Execute**(FrGrMonitorController* Controller)

This member funtion contains all of the functionality needed to process this request.

Protected Functions

FrGrSoftwareMonitor* **CreateSwMonitor**(void)

This member function is called by the Execute operation and creates the FrGrSoftwareMonitor.

FrGrSoftwareMonitor* **FindSwMonitor**(FrGrMonitorController* PassedController)

This member function will find a FrGrSoftwareMonitor object if it has already been created. This ensures that creation of identical FrGrSoftwareMonitor objects will be prevented.

Private Data

RWCString **myStringId**

This attribute identifies a particular StringId associated with this request.

RWCString **mySwSubName**

This attribute identifies a software subsystem associated with a particular process.

FrGrMonitorUsRequest

class **FrGrMonitorUsRequest**

This class will process a request from the String Manager to monitor a particular user station.

Base Classes

public **FrGrUsMonitorRequest**

Public Functions

EcTInt **Execute**(FrGrMonitorController* Controller)

This member funtion contains all of the functionality needed to process this request.

Protected Functions

FrGrUsMonitor* **CreateUsMonitor**(void)

This member function is called by the Execute operation and creates the FrGrUsMonitor.

Private Data

RWCString **myRole**

This attribute identifies whether the User Station is being used for Ground Control or Commanding purposes.

RWCString **myStringId**

This attribute identifies a particular StringId associated with this request.

FrGrResourceMonitor

class **FrGrResourceMonitor**

This base class is a generalization of all of the Hardware and Software Monitor objects.

Public Functions

RWCString **GetStatus**()

This operation will retrieve the Status associated with this Monitor object.

Private Data

UserParameter* **myStatus**

This attribute points to the UserParameter object that contains the status associated with

this Monitor object.

FrGrRtsMonitor

class **FrGrRtsMonitor**

This class contains information on the status of a RTS.

Base Classes

public **FrGrHardwareMonitor**

Private Data

RWSet* **myActiveStringSet**

This attribute points to a RWSet that contains ID's of all of the active strings currently on a RTS.

RWSet* **myBackupStringSet**

This attribute points to a RWSet that contains ID's of all of the backup strings currently on a RTS.

FrGrSoftwareMonitor

class **FrGrSoftwareMonitor**

This class contains information on the status of a software process.

Base Classes

public **FrGrResourceMonitor**

Private Data

EcTInt **myPid**

This attribute identifies a PID associated with a FrGrSoftwareMonitor object.

RWCString **myStringId**

This attribute identifies a string associated with a FrGrSoftwareMonitor object.

RWCString **mySwSubsystemName**

This attribute identifies a software subsystem associated with a particular process. This can be Telemetry, RTContact, Command, or Ground Script.

FrGrStopMonitorSwRequest

class **FrGrStopMonitorSwRequest**

This class processes a request from the String Manager to stop monitoring a software process.

Base Classes

public **FrGrSwMonitorRequest**

Public Functions

EcTInt **Execute**(FrGrMonitorController* Controller)

This member function contains all of the functionality needed to process this request.

Protected Functions

EcTInt **DeleteSwMonitor**(FrGrMonitorController* PassedController)

This member function is called by the Execute operation and deletes a FrGrSoftwareMonitor object.

FrGrSoftwareMonitor **FindSwMonitor**(FrGrMonitorController* PassedController)

This member function will find a FrGrSoftwareMonitor object.

FrGrStopMonitorUsRequest

class **FrGrStopMonitorUsRequest**

This class will process a request from the String Manager to stop monitoring a userstation.

Base Classes

public **FrGrUsMonitorRequest**

Public Functions

EcTInt **Execute**(FrGrMonitorController* Controller)

This member function contains all of the functionality needed to process this request.

Protected Functions

EcTInt **DeleteUsMonitor**(FrGrMonitorController* PassedController)

This member function will delete a UsMonitor object from the PassedController's HwSet.

FrGrStrManResMonIF

class **FrGrStrManResMonIF**

This class is used to receive messages from the WS/RTS RMS subsystem.

Public Construction

FrGrStrManResMonIF(const FrGrStrManResMonIF&)

FrGrStrManResMonIF(const FrGrStrManResMonIF&)

This member function creates a duplicate of this class.

FrGrStrManResMonIF()

This member function is the default constructor for this class.

~FrGrStrManResMonIF()

This member function is the destructor for this class.

Public Functions

FrGrMonitorRequest* **CheckQueue**()

This member function will return a Request object from the Queue.

EcTInt **Initialize**(FrGrController* Controller)

This member function will create a Queue for it's Requests. It will set the String Manager Address as well.

EcTInt **ReceiveRequest**(FrGrMonitorRequest* newRequest)

receiveRequest

This member function will receive a Request from the RTS RMS and put that Request in a Queue.

EcTInt **SendStatus**(FrGrMonitorRequest* sentRequest)

This member function will send a status for the passed Request.

Private Data

FdEvEventLogger* **myEventLogger**

This member variable points to the FdEvEventLogger.

RWSet* **myRequestQueue**

This member variable points to the Queue that contains requests.

Address* **myStrManAddress**

This member variable identifies the Address of the String Manager.

FrGrSwMonitorRequest

class **FrGrSwMonitorRequest**

This class is an aggregation of the software monitoring requests.

Base Classes

public **FrGrMonitorRequest**

Private Data

EcTInt **myPid**

This attribute identifies a PID associated with this request.

RWCString **myStringId**

This attribute identifies a string associated with this request.

FrGrSoftwareMonitor* **mySwMonitor**

This attribute points to a FrGrSoftwareMonitor object associated with this request.

FrGrSwitchMonitorUsRequest

class **FrGrSwitchMonitorUsRequest**

This class processes a request by the String Manager to stop monitoring one userstation and start monitoring another. This is necessary when Ground Control or Command Authority changes.

Base Classes

public **FrGrUsMonitorRequest**

Public Functions

EcTInt **Execute**(FrGrController* Controller)

This member function contains all of the functionality needed to process this request.

Private Data

RWCString **myNewHwId**

This attribute identifies the HwId of the userstation that has taken Ground Control or Command Authority.

RWCString **myRole**

This attribute identifies whether the userstations involved have or had Ground Control or Command Authority.

RWCString **myStringId**

This attribute identifies the string associated with this request.

FrGrUsMonitor

class **FrGrUsMonitor**

This class contains information on the status of a userstation.

Base Classes

public **FrGrHardwareMonitor**

Private Data

RWCString **myRole**

This attribute identifies whether the userstation is being used for Ground Control or Command Authority.

RWCString **myStringId**

This attribute identifies a string associated with a userstation.

FrGrUsMonitorRequest

class **FrGrUsMonitorRequest**

This class is a generalization of the requests that affect a FrGrUsMonitor object.

Base Classes

public **FrGrMonitorRequest**

Protected Functions

FrGrUsMonitor* **FindUsMonitor**(FrGrMonitorController* PassedController)

This member function will find a FrGrUsMonitor object.

Private Data

RWCString **myHwId**

This attribute identifies the HwId of the User Station that is affected by this request.

FrGrUsMonitor* **myUsMonitor**

This attribute points to a FrGrUsMonitor object associated with this request.

3.4 Resource Management Subsystem Performance

The most compelling performance requirement that the Resource Management Subsystem must satisfy is that of failure recovery. According to the Flight Operations Segment (FOS) Requirements Specification for the ECS Project, Volume 1: General Requirements (CDRL number 304-CD-001-002), the Resource Management Subsystem is required to recover from a hardware or software component failure within the mission critical processing string within one (1) minute. The FOS design goal for this recovery is thirty (30) seconds.

To satisfy this requirement, the real-time architecture supports the concept of logical strings that run in a "hot backup" mode in order to facilitate an automated failover from one string of components to another. This means that the RMS will provide the Flight Operations Team with the ability to request allocation of hardware and software resources to act in a backup capacity for like components that are actively supporting a spacecraft contact. The RMS will ensure that the ground configuration of the active and backup strings is synchronous. If the backup string is needed, a single user directive entered by an operator with the ground control privilege will begin the failover process. From this point the failover is executed by the RMS software that ensures that there is no more that one active logical string supporting the same activity at one time. Thus, another RMS requirement to ensure "single point of command" is also satisifed.

Much design discussion was dedicated to the issue of automatic versus automated failover. Ultimately the requirements support an automated failover procedure that is set in motion by an operator with privilege within the EOC.

Abbreviations and Acronyms

AGS	ASTER Ground System
AM	Morning (ante meridiem) -- see EOS AM
ANA	Analysis Subsystem
ASTER	Advanced Spaceborne Thermal Emission and Reflection Radiometer (formerly ITIR)
ATC	Absolute Time Command
BAP	Baseline Activity Profile
CA	Command Authority
CAC	Command Activity Controller
CAid	Command Authority Identifier
CAwsID	Command Authority Workstation Identifier
CCSDS	Consultative Committee for Space Data Systems
CERES	Clouds and Earth's Radiant Energy System
CI	Configuration item
CLCW	Command Link Control Words
CMD	Command Subsystem
CMS	Command Management System
CODA	Customer Operations Data Accounting
COTS	Commercial Off-The-Shelf
CSCI	Computer software configuration item
CSMS	Communications and System Management Segment
CSS	Communications Subsystem (CSMS)
DAR	Data Acquisition Request
DAS	Detailed Activity Schedule
DB	Database
DbId	Database Identifier
DBMS	Database Management System
Decomm	Decommutation
DFCD	Data Format Control Document
DID	Data item description; data ingest/distribution
DMS	Data Management Subsystem
DSN	Deep Space Network
DSS	Decision Support System

ECL	ECS Command Language
ECOM	EOS Communications
ECS	EOSDIS Core System
EDOS	EOS Data and Operations System
EDU	EDOS Data Unit
EOC	EOS Operations Center
EOS	Earth Observing System
EOSDIS	EOS Data and Information System
FDF	Flight Dynamics Facility
FIFO	First In - First Out
FOP	Frame Operation Procedure
FOS	Flight Operations Segment (ECS)
FOT	Flight Operations Team
FSE	FOT S/C Evolutions
FUI	FOS User Interface Subsystem
GCMR	Ground Control Message Request
GSC	Ground Script Controller
HK	Housekeeping
HS	Health and Safety
HW	Hardware
HwId	Hardware Identifier
I&T	Integration and Test
ICC	Instrument Control Center
ID	Identifier
IF	Interface
IP	International Partners
IRD	Interface requirements document
IST	Instrument Support Toolkit
JPL	Jet Propulsion Laboratory
LAN	Local Area Network
LaRC	Langley Research Center
LMC	Lockheed Martin Corporation
LSM	Local System Manager
LTIP	Long Term Instrument Plan
LTSP	Long Term Science Plan
MISR	Multi-Angle Imaging SpectroRadiometer

MO&DSD	Mission Operations and Data Systems Directorate (GSFC Code 500)
MODIS	Moderate Resolution Imaging Spectrometer
MOPITT	Measurements of Pollution in the Troposphere
MSS	CSMS Management Subsystem
MSS	Managememt and Subsystem (part of CSMS)
MTPE	Mission to Planet Earth
Nascom	NASA Communications Network
NASDA	National Space Development Agency (Japan)
NCC	Network Control Center
NOAA	National Oceanic and Atmospheric Administration
OASIS	Operations and Science Instrument Support
OMT	Object Model Technique
OOD	Object Oriented Design
PAS	Planning and Scheduling
PDB	Project Data Base
PI	Principal Investigator
PI/TL	Principal Investigator/Team Leader
PID	Process Identifier
PS	Parameter Server Subsystem
RCM	Real-Time Contact Management Subsystem
RMA	Reliability, Maintainability, Availability
RMS	Resource Management Subsystem
RT	Real Time
RTCS	Relative Time Command Sequence
RTS	Real Time Servers, Relative Time Sequence
RW	Rogue Wave
RWC	Rogue Wave Class
SB	Standby
SCC	Spacecraft Controls Computer
SCDO	Science & Communication Data Operation
SMC	Service Management Center
SN	Space Network
SSIM	Spacecraft Simulator
SSR	Solid State Recorder
STOL	System Test and Operations Language
SW	Software

TD	Target Day
TDRS	Tracking and Data Relay Satellite
TDRSS	Tracking and Data Relay Satellite System
TL	Team Leader
TLM	Telemetry Subsystem
TOO	Target of Opportunity
TW	Target Week
UI	User Interface
UTC	Universal Time Coordinated
WAN	Wide Area Network
WOTS	Wallops Orbital Tracking Station
WS	Workstation

Glossary

GLOSSARY of TERMS for the Flight Operations Segment

activity	A specified amount of scheduled work that has a defined start date, takes a specific amount of time to complete, and comprises definable tasks.
analysis	Technical or mathematical evaluation based on calculation, interpolation, or other analytical methods. Analysis involves the processing of accumulated data obtained from other verification methods.
attitude data	Data that represent spacecraft orientation and onboard pointing information. Attitude data includes: • Attitude sensor data used to determine the pointing of the spacecraft axes, calibration and alignment data, Euler angles or quaternions, rates and biases, and associated parameters. • Attitude generated onboard in quaternion or Euler angle form. • Refined and routine production data related to the accuracy or knowledge of the attitude.
availability	A measure of the degree to which an item is in an operable and committable state at the start of a "mission" (a requirement to perform its function) when the "mission" is called for an unknown (random) time. (Mathematically, operational availability is defined as the mean time between failures divided by the sum of the mean time between failures and the mean down time [before restoration of function].

availability (inherent) (A_i)

The probability that, when under stated conditions in an ideal support environment without consideration for preventive action, a system will operate satisfactorily at any time. The "ideal support environment" referred to, exists when the stipulated tools, parts, skilled work force manuals, support equipment and other support items required are available. Inherent availability excludes whatever ready time, preventive maintenance downtime, supply downtime and administrative downtime may require. A_i can be expressed by the following formula:

$$A_i = MTBF/ (MTBF + MTTR)$$

Where: MTBF = Mean Time Between Failures

MTTR = Mean Time To Repair

availability (operational) (A_o)

The probability that a system or equipment, when used under stated conditions in an actual operational environment, will operate satisfactorily when called upon. A_o can be expressed by the following formula:

$$A_o = MTBM / (MTBM + MDT + ST)$$

Where: MTBM = Mean Time Between Maintenance (either corrective or preventive)

MDT = Mean Maintenance Down Time where corrective, preventive administrative and logistics actions are all considered.

ST = Standby Time (or switch over time)

baseline activity profile

A schedule of activities for a target week corresponding to normal instrument operations constructed by integrating long term plans (i.e., LTSP, LTIP, and long term spacecraft operations plan).

build

An assemblage of threads to produce a gradual buildup of system capabilities.

calibration	The collection of data required to perform calibration of the instrument science data, instrument engineering data, and the spacecraft engineering data. It includes pre-flight calibration measurements, in-flight calibrator measurements, calibration equation coefficients derived from calibration software routines, and ground truth data that are to be used in the data calibration processing routine.
command	Instruction for action to be carried out by a space-based instrument or spacecraft.
command authority	A privilege or designation bestowed on EOC operators to act in critical roles within the EOC. Command authority is granted to one EOC user per command destination for the purpose of sending real-time commands to a spacecraft. This privilege is managed within a logical string to ensure that there is a single point of command for an EOC spacecraft.
command and data handling (C&DH)	The spacecraft command and data handling subsystem which conveys commands to the spacecraft and research instruments, collects and formats spacecraft and instrument data, generates time and frequency references for subsystems and instruments, and collects and distributes ancillary data.
command group	A logical set of one or more commands which are not stored onboard the spacecraft and instruments for delayed execution, but are executed immediately upon reaching their destination on board. For the U.S. spacecraft, from the perspective of the EOS Operations Center (EOC), a preplanned command group is preprocessed by, and stored at, the EOC in preparation for later uplink. A real-time command group is unplanned in the sense that it is not preprocessed and stored by the EOC.
data source	One of five attributes of a logical string that makes it unique. The data source is an enumerated type that indicates the origin of the telemetry data being monitored within the logical string (i.e., real-time, simulation, historical replay)
dedicated service	(aka dedicated logical string)A service dedicated to a single user whereas resources that reside only on the requesting user's workstation are employed to provide the requested service.

detailed activity schedules	The schedule for a spacecraft and instruments which covers up to a 10-day period and is generated/updated daily based on the instrument activity listing for each of the instruments on the respective spacecraft. For a spacecraft and instrument schedule the spacecraft subsystem activity specifications needed for routine spacecraft maintenance and/or for supporting instruments activities are incorporated in the detailed activity schedule.
direct broadcast	Continuous down-link transmission of selected real-time data over a broad area (non-specific users).
EOS Data and Operations System (EDOS) production data set	Data sets generated by EDOS using raw instrument or spacecraft packets with space-to-ground transmission artifacts removed, in time order, with duplicate data removed, and with quality/ accounting (Q/A) metadata appended. Time span or number of packets encompassed in a single data set are specified by the recipient of the data. These data sets are equivalent to Level 0 data formatted with Q/A metadata. For EOS, the data sets are composed of: instrument science packets, instrument engineering packets, spacecraft housekeeping packets, or onboard ancillary packets with quality and accounting information from each individual packet and the data set itself and with essential formatting information for unambiguous identification and subsequent processing.
failure recovery	The process by which the RMS acts upon an operator request to transfer active processing and control of an EOS spacecraft from one string of EOC hardware and software resources to another.
ground control	A privilege or designation that is granted to one EOC user per logical string for the purpose of modifying the ground configuration of the hardware and software resources within that logical string.
historical replay logical string	A logical string with a data source of the replay type. This logical string requires the support of only real-time Telemetry processes for the purpose of decommutating the historical data.
housekeeping data	The subset of engineering data required for mission and science operations. These include health and safety, ephemeris, and other required environmental parameters.

instrument	• A hardware system that collects scientific or operational data. • Hardware-integrated collection of one or more sensors contributing data of one type to an investigation. • An integrated collection of hardware containing one or more sensors and associated controls designed to produce data on/in an observational environment.
instrument activity deviation list	An instrument's activity deviations from an existing instrument activity list, used by the EOC for developing the detailed activity schedule.
instrument activity list	An instrument's list of activities that nominally covers seven days, used by the EOC for developing the detailed activity schedule.
instrument engineering data	Subset of telemetered engineering data required for performing instrument operations and science processing
instrument microprocessor memory loads	Storage of data into the contents of the memory of an instrument’s microprocessor, if applicable. These loads could include microprocessor-stored tables, microprocessor-stored commands, or updates to microprocessor software.
instrument resource deviation list	An instrument's anticipated resource deviations from an existing resource profile, used by the EOC for establishing TDRSS contact times and building the preliminary resource schedule.
instrument resource profile	Anticipated resource needs for an instrument over a target week, used by the EOC for establishing TDRSS contact times and building the preliminary resource schedule.
instrument science data	Data produced by the science sensor(s) of an instrument, usually constituting the mission of that instrument.
logical string	A collection of hardware and software resources, and information about how those resources are being used within the EOC, to provide spacecraft and instrument control and monitoring during real-time contacts, simulations, and historical replays.
long-term instrument plan (LTIP)	The plan generated by the instrument representative to the spacecraft's IWG with instrument-specific information to complement the LTSP. It is generated or updated approximately every six months and covers a period of up to approximately 5 years.

long-term science plan (LTSP)	The plan generated by the spacecraft's IWG containing guidelines, policy, and priorities for its spacecraft and instruments. The LTSP is generated or updated approximately every six months and covers a period of up to approximately five years.
long term spacecraft operations plan	Outlines anticipated spacecraft subsystem operations and maintenance, along with forecasted orbit maneuvers from the Flight Dynamics Facility, spanning a period of several months.
mean time between failure (MTBF)	The reliability result of the reciprocal of a failure rate that predicts the average number of hours that an item, assembly or piece part will operate within specific design parameters. (MTBF=1/(l) failure rate; (l) failure rate = # of failures/ operating time.
mean down time (MDT)	Sum of the mean time to repair MTTR plus the average logistic delay times.
mean time between maintenance (MTBM)	The mean time between preventive maintenance (MTBPM) and mean time between corrective maintenance (MTBCM) of the ECS equipment. Each will contribute to the calculation of the MTBM and follow the relationship: 1/MTBM = 1/MTBPM + 1/MTBCM
mean time to repair (MTTR)	The mean time required to perform corrective maintenance to restore a system/equipment to operate within design parameters.
mirrored connection	(aka mirrored logical string connection) Type of connection to a shared service in which the requesting user is provided the same telemetry ground configuration as the telemetry processes running on the Real-Time Server. See also tailored connection.
mission critical	A term used to describe an activity, function or EOC resource that provides a service that is necessary for ensuring the well-being of an EOS spacecraft.
mode	One of five attributes of a logical string that makes it unique. The mode is an enumerated type which indicates the operator's intended use of the logical string (i.e., operational, test, training)
object	Identifiable encapsulated entities providing one or more services that clients can request. Objects are created and destroyed as a result of object requests. Objects are identified by client via unique reference.

operational database identifier	One of five attributes of a logical string that makes it unique. The operational database id indicates which database version is used in configuration of the software in a specific logical string
orbit data	Data that represent spacecraft locations. Orbit (or ephemeris) data include: Geodetic latitude, longitude and height above an adopted reference ellipsoid (or distance from the center of mass of the Earth); a corresponding statement about the accuracy of the position and the corresponding time of the position (including the time system); some accuracy requirements may be hundreds of meters while other may be a few centimeters.
permanent process	(aka persistent process) Software process (task) that is executed upon host startup and terminated upon host shutdown. See also transient process.
playback data	Data that have been stored on-board the spacecraft for delayed transmission to the ground.
preliminary resource schedule	An initial integrated spacecraft schedule, derived from instrument and subsystem resource needs, that includes the network control center TDRSS contact times and nominally spans seven days.
preplanned stored command	A command issued to an instrument or subsystem to be executed at some later time. These commands will be collected and forwarded during an available uplink prior to execution.
principal investigator (PI)	An individual who is contracted to conduct a specific scientific investigation. (An instrument PI is the person designated by the EOS Program as ultimately responsible for the delivery and performance of standard products derived from an EOS instrument investigation.).
prototype	Prototypes are focused developments of some aspect of the system which may advance evolutionary change. Prototypes may be developed without anticipation of the resulting software being directly included in a formal release. Prototypes are developed on a faster time scale than the incremental and formal development track.

raw data	Data in their original packets, as received from the spacecraft and instruments, unprocessed by EDOS. • Level 0 – Raw instrument data at original resolution, time ordered, with duplicate packets removed. • Level 1A – Level 0 data, which may have been reformatted or transformed reversibly, located to a coordinate system, and packaged with needed ancillary and engineering data. • Level 1B – Radiometrically corrected and calibrated data in physical units at full instrument resolution as acquired. • Level 2 – Retrieved environmental variables (e.g., ocean wave height, soil moisture, ice concentration) at the same location and similar resolution as the Level 1 source data. • Level 3 – Data or retrieved environmental variables that have been spatially and/or temporally resampled (i.e., derived from
real-time data	Data that are acquired and transmitted immediately to the ground (as opposed to playback data). Delay is limited to the actual time required to transmit the data.
real-time logical strings	A logical string with a real-time data source. These logical string are nominally used to operationally monitor real-time contacts, and require the support of a full complement of Telemetry, Command and Real-time Contact Management subsystem processes as well as FUI Ground Script Controller, DMS Archiver and Parameter Server processes.
reconfiguration	A change in operational hardware, software, data bases or procedures brought about by a change in a system's objectives.
SCC-stored commands and tables	Commands and tables which are stored in the memory of the central onboard computer on the spacecraft. The execution of these commands or the result of loading these operational tables occurs sometime following their storage. The term "core-stored" applies only to the location where the items are stored on the spacecraft and instruments; core-stored commands or tables could be associated with the spacecraft or any of the instruments.
scenario	A description of the operation of the system in user's terminology including a description of the output response for a given set of input stimuli. Scenarios are used to define operations concepts.

segment	One of the three functional subdivisions of the ECS: CSMS -- Communications and Systems Management Segment FOS -- Flight Operations Segment SDPS -- Science Data Processing Segment
sensor	A device which transmits an output signal in response to a physical input stimulus (such as radiance, sound, etc.). Science and engineering sensors are distinguished according to the stimuli to which they respond. • Sensor name: The name of the satellite sensor which was used to obtain that data.
shared service	(aka shared logical string) A logical string created to provide a specific service to multiple users. Resources are allocated on a Real-Time Server and multiple users from different user workstations are allowed to individually access the shared service and monitor the same activity.
spacecraft identifier	One of five attributes of a logical string that makes it unique. The spacecraft id marks a logical string for support of a specific mission.
spacecraft engineering data	The subset of engineering data from spacecraft sensor measurements and on-board computations.
spacecraft subsystems activity list	A spacecraft subsystem's list of activities that nominally covers seven days, used by the EOC for developing the detailed activity schedule.
spacecraft subsystems resource profile	Anticipated resource needs for a spacecraft subsystem over a target week, used by the EOC for establishing TDRSS contact times and building the preliminary resource schedule.
state	one of five attributes of a logical string that makes it unique. The state is an enumerated type that indicates if a logical string is actively performing its intended function (active), or if it exists only to perform its intended function in the event of a hardware or software failure in the active string (backup)
tailored connection	(aka tailored logical string connection) Type of connection to a shared service in which the requesting user is allowed to modify the ground configuration of the telemetry processes running on the local workstation, independent of the telemetry processes executing on the Real-Time Server. The user may tailor the ground configuration of the local processing to his own needs. See also mirrored connection.

target of opportunity (TOO)	A TOO is a science event or phenomenon that cannot be fully predicted in advance, thus requiring timely system response or high-priority processing.
thread	A set of components (software, hardware, and data) and operational procedures that implement a function or set of functions.
thread, *as used in some Systems Engineering documents*	A set of components (software, hardware, and data) and operational procedures that implement a scenario, portion of a scenario, or multiple scenarios.
toolkits	Some user toolkits developed by the ECS contractor will be packaged and delivered on a schedule independent of ECS releases to facilitate science data processing software development and other development activities occurring in parallel with the ECS.
Transient Process	(aka temporary process) Software process (task) that is executed by a parent process in order to perform a specific function. Upon completion of that function, the process is terminated by the parent process. See also permanent processs.